FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Interner Bericht

Implementierung eines parallelen Gleichungslösers in das Finite-Elemente-Programm FINEART

Henning Niehoff

FZJ-ZAM-IB-2002-05

1. Auflage

(letzte Änderung: 23. Mai 2002)

Zusammenfassung

Implementierung eines parallelen Gleichungslösers in das Finite-Elemente-Programm FINEART

Die Finite-Elemente-Methode (FEM) ist ein Verfahren zur numerischen Lösung von partiellen Differentialgleichungen. Die Genauigkeit einer FEM-Berechnung ist vor allem von der Güte des Berechnungsnetzes abhängig. Ein entscheidendes Kriterium hierfür ist die Feinheit des Netzes, das im Allgemeinen mit Hilfe von Netzgeneratoren konstruiert wird. Da die numerischen Fehler nur schwer vorhersehbar sind, ist die Qualität des Netzes von der Vorgehensweise des Anwenders bzw. Ingenieurs und dessen Erfahrung bestimmt. Daher werden seit einiger Zeit Algorithmen zur automatischen Netzverfeinerung durch Einbindung zusätzlicher finite Elemente entwickelt.
FINEART ist ein FEM-Programm, das zusätzlich zur herkömmlichen FEM-Berechnung Module zur automatischen Fehlerbestimmung und adaptiven Netzverfeinerung beinhaltet. Im Rahmen eines Projekts der Forschungszentrum Jülich GmbH mit dem Structural Engineering Research Centre (SERC), Indien, entstand eine Diplomarbeit, in der untersucht wurde, in welchem Maße sich die Effizienz des Lösungsmoduls von FINEART durch Implementierung eines parallelen Gleichungslösers steigern lässt. Dazu wurde ein neues Lösungsmodul implementiert, das parallele Gleichungslöser der Programmbibliothek ScaLAPACK verwendet. Die Effizienz dieser Löser bzw. des neuen Moduls wurden auf dem SMP-Cluster ZAMpano bzw. dem massiv parallelen System CRAY T3E untersucht und mit dem bisher in FINEART genutzten seriellen Löser verglichen.

Implementation of a parallel linear equation solver into the finite element program FINEART

The finite element method (FEM) is a procedure to solve partial differential equations numerically. The accuracy of a FEM calculation depends mainly on the quality of the computation mesh. An important criterion is the fineness of the mesh which is generally constructed with mesh generators. Since the numerical errors are difficult to predict, the quality of the mesh is determined by the strategy of the user resp. engineer and his experience. That is the reason why algorithms for automatic adaptive mesh refinement by integrating additional finite elements have been developed.
FINEART is a FEM program that includes, in addition to conventional FEM calculation, modules for automatic error estimation and adaptive refinement. Within a project of the Research Centre Jülich and the Structural Engineering Research Centre (SERC), India, a diploma thesis emerged, studying how the efficiency of the FINEART solver module can be enhanced by implementing a parallel solver. Therefore a new solver module, which uses parallel linear equation solvers of the program library ScaLAPACK has been implemented. The efficiency of these solvers resp. the new module was examined on the SMP-cluster ZAMpano resp. the massively-parallel system CRAY T3E and compared with the serial solver that has been used in FINEART up to now.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Es gibt eine Reihe von Näherungsverfahren zur Berechnung komplexer Probleme, die nicht exakt gelöst werden können. Sie werden beispielsweise zur Simulation von Crash-Tests in der Automobilindustrie und zur Untersuchung von Statikverhalten von Bauwerken bei Erdbeben angewandt. Teilweise können sie aber auch für ganz andere Probleme, wie z.B. bei der Wettervorhersage genutzt werden. Dabei wird die Finite-Elemente-Methode (FEM) besonders häufig eingesetzt (Abb. 1.1). Vor der Anwendung muss das Problem zunächst entsprechend untersucht werden. Die prinzipielle Vorgehensweise ähnelt dabei sehr anderen ingenieurmäßigen Methoden. Das sehr spezielle Einzelproblem wird solange vereinfacht und verallgemeinert, bis ein überschaubares Modell entsteht, das mit bekannten, allgemein gültigen und anerkannten Ansätzen bearbeitet werden kann. Dazu werden meist Computerprogramme als Hilfe benutzt. Ein vollständiges FEM-Programmpaket besteht dabei aus drei Komponenten: Der Preprozessor unterstützt den Anwender bei der Modellbildung, das dann von einem Löser (Solver) berechnet wird. Die Ergebnisse können anschließend mit dem Postprozessor dargestellt und überprüft werden. Bei einer ungenügenden Genauigkeit der Ergebnisse muss eine Anpassung des Modells erfolgen.

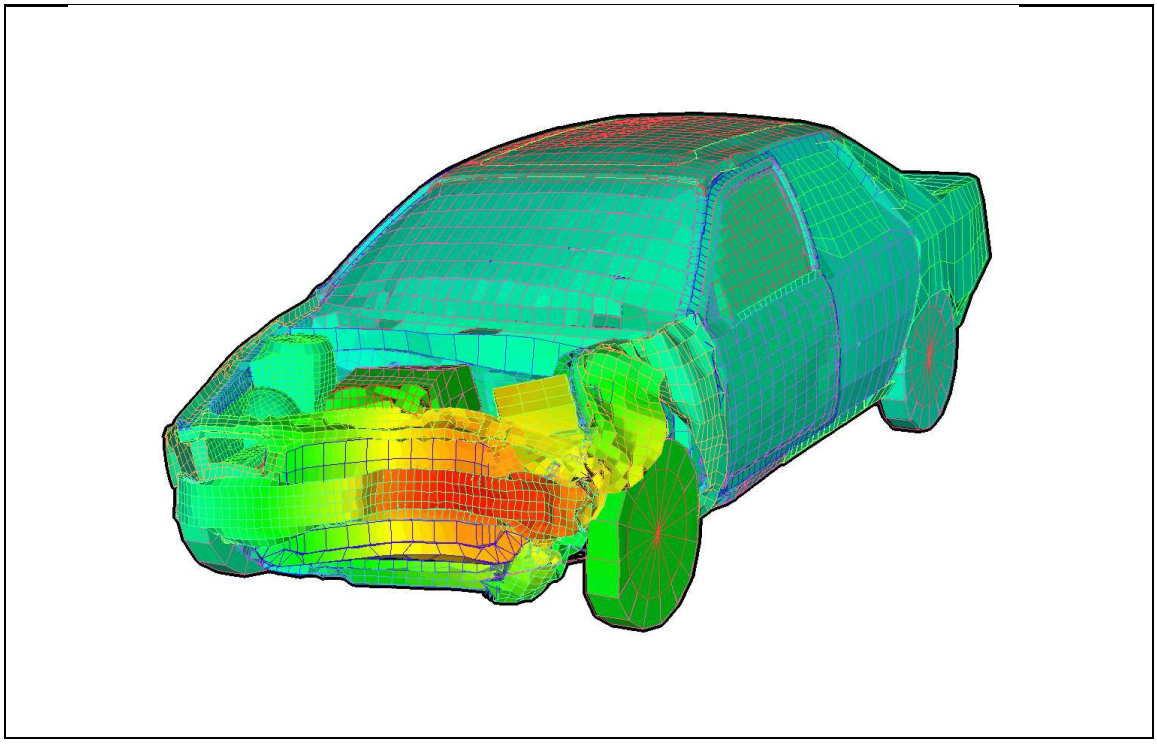


Abbildung 1.1: FEM-Berechnungsnetz eines PKW

Diese Diplomarbeit beschäftigt sich mit der Implementierung eines parallelen Gleichungslösers aus einer vorhandenden Programmbibliothek in das FEM-Programm FINEART. Dieses Programm dient insbesondere zur Berechnung und anschließenden Optimierung eines FEM-Modells. So genügt bei der Modellerstellung ein eher grobes FEM-Ausgangsnetz, das dann von FINEART berechnet und automatisch iterativ verfeinert wird, bis die Lösung eine gewünschte Genauigkeit erreicht. Das Ziel der Implementierung ist eine effizientere Berechnung eines FEM-Problems durch einen Gleichungslöser, der mehrere Prozessoren nutzt.

Der Aufbau dieser Arbeit gestaltet sich dabei folgendermaßen:
In Kapitel 2 werden alle wichtigen Grundlagen vermittelt. Dabei wird noch einmal detailliert auf die FEM eingegangen. Ebenso wird das, zum Lesen dieser Arbeit, notwendige Wissen über die parallele Datenverarbeitung beschrieben. Dabei werden insbesondere die verwendeten Systeme ZAMpano (ZAM parallel nodes) und CRAY T3E des Zentralinstituts für Angewandte Mathematik (ZAM), Forschungszentrum Jülich GmbH vorgestellt.
Kapitel 3 gibt eine allgemeine Beschreibung über die Programmbibliothek ScaLAPACK, die einige parallele Gleichungslöser zur Verfügung stellt. Dabei wird die Funktionsweise zweier Löser erklärt sowie alle notwendigen Schritte, die beim Einsatz zu beachten sind. Des Weiteren werden erste, von FINEART unabhängige, Messungen zur Rechnenzeit durchgeführt und diskutiert.
In Kapitel 4 wird die Funktion von FINEART selbst, u.a. anhand eines Beispiels, erläutert. Dabei werden die entsprechenden Ergebnisse mit dem Postprozessor RAPS dargestellt und erörtert. Anschließend werden, wie im vorigen Kapitel, Messungen durchgeführt, die diesmal FINEART spezifisch sind. Die Ergebnisse zwischen dem, von FINEART ursprünglich verwendeten, seriellen Löser und dem neu implementierten parallelen Löser werden miteinander verglichen.
Als letztes wird noch einmal eine Zusammenfassung aller Resultate sowie ein Ausblick gegeben.

Kapitel 2

Grundlagen

2.1 Die Finite-Elemente-Methode

In Industrie und Forschung müssen bei der Entwicklung von Konstruktionen, Geräten oder Bauteilen verschiedene Faktoren berücksichtigt werden. Zum einen soll eine gewisse Sicherheit und Verfügbarkeit während der gesamten Betriebszeit gewährleistet sein. Andererseits soll die Herstellung wirtschaftlich, d.h. kostengünstig und zeitsparend sein. Aufgabe des Ingenieurs ist es, eine möglichst optimale Lösung dieses sogenannten **Festigkeitsproblems** [1] zu finden. Dazu muss insbesondere untersucht werden, wie belastbar das System ist und welche Auswirkungen bei Versagen entstehen bzw. zu welcher Versagensart es kommen kann. Dabei wird zwischen unzulässig großen Verformungen (elastisch, plastisch), Brüchen und Instabilitäten (Kippen, Knicken, Beulen) unterschieden. Die dafür entscheidenden Größen sind die Verschiebungen und vorliegenden Spannungen (Kap. 2.1.3) einzelner Strukturteile. Um kostensparend zu arbeiten, werden unnötige Experimente vermieden und in der Planungsphase entsprechende Analysen durchgeführt. Das Objekt wird dafür so weit vereinfacht, dass bekannte Rechenansätze und Methoden, z.B. aus der Technischen Mechanik [2], angewandt werden können. Dabei muss darauf geachtet werden, dass die Ergebnisse der Modellrechnung das reale Problem hinreichend genau beschreiben.
Bei der Erstellung eines geeigneten Modells werden drei Typen von Informationen benötigt: Die Geometrie beschreibt die Form und die Abmessungen des zu analysierenden Bauteils. Die Werkstoffeigenschaften bezeichnen die Gesamtheit der inneren Gesetze der Stoffe, aus denen die Struktur besteht. Sie werden durch entsprechende Parameter charakterisiert. Randbedingungen sind die äußeren Einflüsse der Umgebung. Für alle drei Typen werden nur notwendige Informationen übernommen, die für die Berechnung relevant sind.
Das so vorliegende Objekt besteht aus einer oder mehreren zusammenhängenden Teilstrukturen, die als **Kontinuum** betrachtet werden, d.h. diese Teilstrukturen bilden kontinuierliche, fortgesetzte Bereiche, die nur wenige ausgezeichnete Punkte besitzen. Anschließend wird das Kontinuum durch ein Netz oder Gitter diskretisiert. An ausgewählten Punkten des Gebietes werden die bedeutsamen Größen, wie Verschiebungen, Dehnungen und Spannungen, durch Näherungsverfahren berechnet. Eine Gesamtlösung ergibt sich dann durch Interpolation zwischen den Gitterpunkten.
Die ständig steigende Leistungsfähigkeit von EDV-Systemen in den letzten Jahren hat dazu geführt, dass vielfältige Möglichkeiten entstanden sind, technische Probleme mit Hilfe von Simulationsverfahren in relativ kurzer Zeit zu lösen. Die Bedeutung dieser Verfahren wird in Zukunft weiter zunehmen, denn sie werden immer noch weiterentwickelt und die Steigerung der Rechenleistung der dafür nötigen Computerhardware, bei gleichbleibenden Preis, scheint unaufhörlich zu sein.

Ein in diesem Gebiet häufig eingesetztes Verfahren zur Berechnung komplexer Strukturen ist die Finite-Elemente-Methode [3][4][5][6][7]. Dabei werden die physikalischen Größen des Kontinuums durch partielle Differentialgleichungen [8] beschrieben. Um diese berechnen zu können, müssen

diejenigen Funktionen bestimmt werden, die die Differentialgleichungen einschließlich der gestellten Rand- (und Anfangs-) bedingungen erfüllen. Da dies nur selten exakt gelingt, werden diese durch Ansatz- bzw. Formfunktionen approximiert. Zur näherungsweisen Berechnung wird das Kontinuum bei der Diskretisierung in kleine Teilgebiete mit möglichst einfacher Struktur unterteilt, die finiten Elemente. Beschränkt man sich auf zweidimensionale Probleme, bieten sich Drei- oder Vierecke als Elemente an; bei der Erweiterung zur dritten Dimension würde dies zu Tetraeder- bzw. Hexaeder-Elementen führen. Die Elemente werden so zusammengesetzt, dass für die Berechnungen ein brauchbares Modell entsteht.
Die Qualität der Lösung hängt im Wesentlichen von der Güte des verwendeten Netzes ab. Es muss also zunächst ein geeignetes Berechnungsnetz generiert werden, das den Bedingungen des später verwendeten FEM-Approximationsverfahrens genügt. Anschließend werden alle Berechnungsgrößen auf Werte in den Gitterpunkten, den sogenannten **Knoten**, zurückgeführt. Bei Festigkeitsproblemen führt dies zu einem Gleichungssystem zwischen den Knotenverschiebungen und den Kräften in den Knoten. Diese werden zunächst unabhängig für die einzelnen Elemente aufgestellt und anschließend zu einem globalen Gleichungssystem miteinander verknüpft.
Werden dann noch die Lager- bzw. Randbedingungen berücksichtigt, so kann das globale Gleichungssystem nach den möglichen Verschiebungen der freien Knoten aufgelöst werden. Die Anzahl der Bewegungsmöglichkeiten eines Knotens werden dabei als **Freiheitsgrade** bezeichnet. Die Interpolation der Gesamtlösung wird über die Formfunktionen durchgeführt.

2.1.1 Berechnung der Verschiebungen

Um die Verschiebungen der Knoten eines finiten Elementes bestimmen zu können, wird bei der konkreten FEM-Berechnung zunächst das folgende Gleichungssystem aufgestellt

$$\boldsymbol{K}_e \vec{d}_e = \vec{f}_e, \tag{2.1}$$

wobei $\boldsymbol{K}_e$ die Element-Steifigkeitsmatrix, $\vec{d}_e$ der Knotenverschiebungsvektor und $\vec{f}_e$ der Kräftevektor in dem jeweiligen Element ist.
Dies entspricht der allgemeinen Standardnotation für lineare Gleichungssysteme (LGS) [9] [10] [11]

$$\boldsymbol{A}\vec{x} = \vec{b} \qquad \Leftrightarrow \qquad \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}, \tag{2.2}$$

wobei $\boldsymbol{A}$ die Koeffizientenmatrix, $\vec{b}$ die Rechte-Hand-Seite (RHS) und $\vec{x}$ der Lösungsvektor ist.
n gibt die Dimension der Matrix an.
Die Knotenverschiebungen $\vec{d}_e$ werden auf die Verschiebungen $\vec{u}_e$ eines beliebigen Punktes eines finiten Elementes mit Hilfe der Ansatz- bzw. Formfunktionen $\boldsymbol{N}$ extrapoliert, so dass gilt

$$\vec{u}_e = \boldsymbol{N}\vec{d}_e. \tag{2.3}$$

Die Ansatzfunktionen sind Polynome, die so konstruiert werden, dass gilt

$$N_i = \begin{cases} 1 & \text{für Knoten } i \\ 0 & \text{für alle anderen Knoten} \end{cases}. \tag{2.4}$$

Bei der Aufstellung des Gleichungssystems (2.1) muss zunächst die Element-Steifigkeitsmatrix $\boldsymbol{K}_e$ bestimmt werden. Dies geschieht über die Ableitungen $\boldsymbol{D}$ der Ansatzfunktionen $\boldsymbol{N}$ durch das Volumenintegral

$$\boldsymbol{K}_e = \int_V \boldsymbol{B}^T \boldsymbol{E} \boldsymbol{B}\, dV \qquad \text{mit} \qquad \boldsymbol{B} = \boldsymbol{D}\boldsymbol{N}, \tag{2.5}$$

wobei $\boldsymbol{B}$ die Dehnungs-Verschiebungsmatrix ist, die für zweidimensionale Probleme gegeben ist durch

$$\boldsymbol{B} = \begin{pmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{pmatrix} \begin{pmatrix} N_1 & 0 & N_2 & 0 & \dots \\ 0 & N_1 & 0 & N_2 & \dots \end{pmatrix}. \tag{2.6}$$

$\boldsymbol{E}$ ist eine materialspezifische, symmetrische Elastizitätsmatrix, die sich aus dem Zusammenhang zwischen Spannung und Dehnung (Kap. 2.1.3, Gl. (2.29)) ergibt. Sie setzt sich aus dem Elastizitätsmodul E und der Poisson-Zahl bzw. Querkontraktionszahl ν zusammen. Im Zweidimensionalen ist $\boldsymbol{E}$ im ebenen Spannungszustand gegeben durch

$$\frac{E}{1-\nu^2} \begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{pmatrix}. \tag{2.7}$$

Da $\boldsymbol{E}$ symmetrisch ist, ist auch die nach Gleichung (2.5) definierte Element-Steifigkeitsmatrix $\boldsymbol{K}_e$ symmetrisch.

Aus den lokalen Elementgrößen werden dann nach dem Superpositions-Prinzip (Überlagerungs-Prinzip) die globalen Größen für das gesamte System unter Ausnutzung der Randbedingungen zusammengesetzt.

$$\boldsymbol{K}\vec{u} = \vec{f} \tag{2.8}$$

Dazu werden die lokalen Element-Gleichungssysteme in den Knotenwerten (Gl. (2.1)) betrachtet und in das globale Gleichungssystem

$$\boldsymbol{K}\vec{d} = \vec{f} \tag{2.9}$$

überführt, womit über die Ansatzfunktionen die Verschiebungen $\vec{u}$ des gesamten Systems ermittelt werden können. Dazu werden Inzidenzmatrizen (Boole'sche Matrizen) $\boldsymbol{A}_e$ konstruiert, die eine Verknüpfung zwischen den verschiedenen lokalen Knotenwerten und den globalen Knotenwerten herstellen.

$$\vec{d_e} = \boldsymbol{A}_e\vec{d} \tag{2.10}$$

Da die Berechnung des Gleichungssystems (2.9) in dieser Arbeit eine besondere Rolle spielt, wird dieser Teil nun ausführlicher beschrieben:
Nach dem Schnittprinzip besteht ein Gleichgewicht zwischen den äußeren Lasten $\vec{f}$ und den Schnittkräften $\vec{f_e}$. Äquivalent dazu gilt nach dem Prinzip der virtuellen Arbeit die Gleichheit zwischen der virtuellen äußeren und inneren Arbeit

$$\underbrace{\delta\vec{d}^T\vec{f}}_{\text{virtuelle äußere Arbeit}} = \underbrace{\sum_{e=1}^{n} \delta\vec{d_e}^T\vec{f_e}}_{\text{virtuelle Arbeit der Schnittkräfte}} \underset{(2.10)}{=} \sum_{e=1}^{n} \delta(\boldsymbol{A}_e\vec{d})^T\vec{f_e} = \delta\vec{d}^T\sum_{e=1}^{n}\boldsymbol{A}_e^T\vec{f_e} \tag{2.11}$$

$$\underset{(2.11)\text{ gilt }\forall\delta\vec{d}\neq\vec{0}}{\Rightarrow} \quad \vec{f} = \sum_{e=1}^{n}\boldsymbol{A}_e^T\vec{f_e}, \tag{2.12}$$

wobei n die Anzahl der Elemente und δ der Variationsoperator ist. Dadurch kann die globale Steifigkeitsmatrix bestimmt werden über

$$\vec{f} = \sum_{e=1}^{n}\boldsymbol{A}_e^T\vec{f_e} \underset{(2.1)}{=} \sum_{e=1}^{n}\boldsymbol{A}_e^T\boldsymbol{K}_e\vec{d_e} \underset{(2.10)}{=} \sum_{e=1}^{n}\boldsymbol{A}_e^T\boldsymbol{K}_e\boldsymbol{A}_e\vec{d} \tag{2.13}$$

$$\Rightarrow \quad \boldsymbol{K} = \sum_{e=1}^{n}\boldsymbol{A}_e^T\boldsymbol{K}_e\boldsymbol{A}_e. \tag{2.14}$$

Das Gleichungssystem (2.9) ist jedoch noch nicht lösbar, da es mehr Unbekannte, als linear unabhängige Gleichungen besitzt. Aus diesem Grund müssen noch die kinematischen Randbedingungen ausgenutzt werden. Ein Teil der Knotenverschiebungen sei vorgegeben, so dass $\vec{d}$ zerlegt werden kann in die vorgegebenen Knotenwerte $\vec{d}_V$ und die freien bzw. gesuchten Knotenwerte $\vec{d}_F$. Entsprechend kann $\vec{f}$ in die äußeren Kräfte $\vec{f}_F$ und die unbekannten Lagerreaktionen $\vec{f}_V$ zerlegt werden. Wird dann noch $\boldsymbol{K}$ in Teilmatrizen unterteilt, führt dies auf das partitionierte Gleichungssystem

$$\begin{pmatrix} \boldsymbol{K}_{VV} & \boldsymbol{K}_{VF} \\ \boldsymbol{K}_{FV} & \boldsymbol{K}_{FF} \end{pmatrix} \begin{pmatrix} \vec{d}_V \\ \vec{d}_F \end{pmatrix} = \begin{pmatrix} \vec{f}_V \\ \vec{f}_F \end{pmatrix}. \tag{2.15}$$

Aus der zweiten Zeile dieses Systems können die freien Knotenverschiebungen berechnet werden durch das reduzierte Gleichungssystem

$$\boldsymbol{K}_{FV}\vec{d}_V + \boldsymbol{K}_{FF}\vec{d}_F = \vec{f}_F \qquad \Leftrightarrow \qquad \vec{d}_F = \boldsymbol{K}_{FF}^{-1}(\vec{f}_F - \boldsymbol{K}_{FV}\vec{d}_V). \tag{2.16}$$

Die erste Zeile des Gleichungssystems (2.15) führt zu den Lagerreaktionen

$$\begin{aligned} \vec{f}_V &= \boldsymbol{K}_{VV}\vec{d}_V + \boldsymbol{K}_{VF}\vec{d}_F \\ &= \boldsymbol{K}_{VV}\vec{d}_V + \boldsymbol{K}_{VF}\boldsymbol{K}_{FF}^{-1}(\vec{f}_F - \boldsymbol{K}_{FV}\vec{d}_V). \end{aligned} \tag{2.17}$$

2.1.2 Eigenschaften der (globalen) Steifigkeitsmatrix

Die (globale) Steifigkeitsmatrix $\boldsymbol{K}$ des Gleichungssystems (2.8) bzw. (2.9) erfüllt in der Regel die folgenden Eigenschaften:

(i) $\boldsymbol{K}$ ist symmetrisch, d.h. $\boldsymbol{K}$ ist gleich der transponierten Matrix $\boldsymbol{K}^T$, also

$$\boldsymbol{K} = \boldsymbol{K}^T \tag{2.18}$$

und damit auch quadratisch, d.h. die Anzahl der Zeilen ist gleich der Anzahl der Spalten. Diese Eigenschaft ergibt sich aus dem Superpositions-Prinzip und der Symmetrie der Element-Steifigkeitsmatrizen.

(ii) $\boldsymbol{K}$ ist positiv definit, für mindestens kinematisch bestimmt gelagerte Systeme, d.h.

$$\vec{x}^T\boldsymbol{K}\vec{x} > 0 \qquad \forall\, \vec{x} \neq \vec{0}. \tag{2.19}$$

Ein System heißt kinematisch bestimmt gelagert, wenn Starrkörperbewegungen unterdrückt werden. Dadurch sind die Lagerreaktionen aus den Gleichgewichtsbedingungen berechenbar. Die Anzahl der Unbekannten muss gleich der Anzahl der Gleichungen sein.
Die globale Steifigkeitsmatrix wurde über das Prinzip der virtuellen Arbeit (Gl.(2.11)) hergeleitet. Da die Arbeit der äußeren Kräfte stets positiv ist, folgt die positive Definitheit.

(iii) $\boldsymbol{K}$ ist dünn besetzt, d.h. die Matrix besteht hauptsächlich aus Nullelementen.
Dies liegt daran, dass in jeder Zeile der Matrix nur die Knotenwerte eines finiten Elementes und die Beiträge der benachbarten Elemente stehen. Demnach wird die relative Anzahl an Nullelementen größer, je mehr finite Elemente verwendet werden.

(iv) $\boldsymbol{K}$ hat Bandstruktur, d.h. es existieren nur Einträge in der Umgebung der Diagonalen. Die Bandbreite wird bestimmt durch den von der Diagonalen am weitesten entfernten Eintrag.
Diese Eigenschaft wird durch eine möglichst effektive Knotennummerierung bei der Vernetzung erzielt. Optimal ist, wenn benachbarte Knoten so nummeriert werden, dass auch deren Nummern möglichst dicht beieinander liegen. Es haben sich verschiedene, unterschiedlich aufwändige, Nummerierungsstrategien entwickelt. Teilweise wird dabei eine eventuell größere Bandbreite in Kauf genommen, um den dafür notwendigen Aufwand geringer zu halten.
Eine Bandmatrix hat die Bandbreite β, falls gilt

$$\forall i,j : |i-j| > \beta \quad : \quad a_{ij} = 0. \tag{2.20}$$

Bei dieser Definition wird die Diagonale selbst nicht mit zur Bandbreite gezählt. In manchen Definitionen gilt dies aber und in Gleichung (2.20) ändert sich das „größer"-Zeichen in ein „größer-gleich"-Zeichen.

(v) Je nach Problem kann $\boldsymbol{K}$ weitere Eigenschaften besitzen, die aber immer spezifischer werden und deshalb nicht weiter behandelt werden.

Zum Lösen von LGS gibt es zum einen direkte Methoden, wie das Gauß'sche Eliminationsverfahren, die abgesehen von numerischen Rundungsfehlern, eine exakte Lösung des LGS liefern. Deren Rechenaufwand ist im Allgemeinen proportional zu n^3. Bei Bandmatrizen verbessert sich dieser auf $n \cdot \beta^2$. Diese Verfahren sind geeignet für kleine bis mittlere Probleme oder schlanke Strukturen, wodurch die Bandbreite in der Matrix verkleinert wird. Der Vorteil ist, dass es einfach ist, mehrere Lastfälle zu bearbeiten, indem die RHS entsprechend erweitert wird. In dieser Arbeit wird nur der Rechenaufwand des eigentlichen Lösungsalgorithmus, unabhängig von der Anzahl an Lastfällen, untersucht. Aus diesem Grund wird nur mit jeweils einem Lastfall gerechnet.
Zum anderen gibt es iterative Verfahren, die nur eine Näherung der exakten Lösung liefern. Dabei wird eine bestimmte Rechenvorschrift mehrmals wiederholt. Dieser Vorgang wird **Iteration** genannt. Im Idealfall nähert sich die berechnete Lösung mit jedem Iterationsschritt der exakten Lösung an (Konvergenz). Die Rechenzeit ist allerdings nicht genau vorhersehbar. Der Aufwand je Iterationsschritt ist im Allgemeinen proportional zu n^2 [12]. Dafür benötigen diese Verfahren weniger Speicher und sind gut nutzbar für große Probleme oder sperrige Strukturen, wodurch die Bandbreite in der Matrix vergrößert wird. In diesem Fall konvergieren diese Methoden schneller gegen die exakte Lösung. Dies ist allerdings abhängig von den Einträgen in der Matrix, da das Verfahren bei einer schlecht konditionierten Matrix numerisch instabil wird. Die Konditionszahl κ einer Matrix A kann berechnet werden über das Produkt der Normen

$$\kappa(A) = \|A\| \cdot \|A^{-1}\|. \tag{2.21}$$

Die Berechnung muss für jeden Lastfall neu gemacht werden.

2.1.3 Spannungen (und Dehnungen)

Wird ein Körper auf Zug oder Druck belastet, entstehen die für spätere Betrachtungen interessanten Spannungen. Sie sind innere Kräfte, die bei einem Schnitt durch den Körper bestimmbar werden. Spannungen sind auf der Schnittfläche verteilte Flächenkräfte. Bei einfachen Strukturen, wie eindimensionalen Stäben, die nur in Längsrichtung auf Zug oder Druck beansprucht werden, ist die Spannung σ definiert durch Kraft F pro Fläche A.

$$\sigma = \frac{F}{A} \tag{2.22}$$

Die Spannungen stehen dabei senkrecht zur Schnittfläche. Aus diesem Grund werden sie Normalspannungen genannt.
Bei einem Körper der beliebig geformt oder belastet ist, z.B. durch Einzelkräfte F_i, sind die Spannungen im Allgemeinen über die Schnittfläche A veränderlich. Daher muss die Spannung in einem beliebigen Punkt P der Schnittfläche definiert werden. Auf einem Flächenelement ΔA, in dem P enthalten ist, wirkt die mittlere Spannung $\Delta\vec{F}/\Delta A$. Wird vorausgesetzt, dass dieser Quotient gegen einen endlichen Wert strebt, gilt

$$\vec{t} = \lim_{\Delta A \to 0} \frac{\Delta \vec{F}}{\Delta A} = \frac{d\vec{F}}{dA}. \tag{2.23}$$

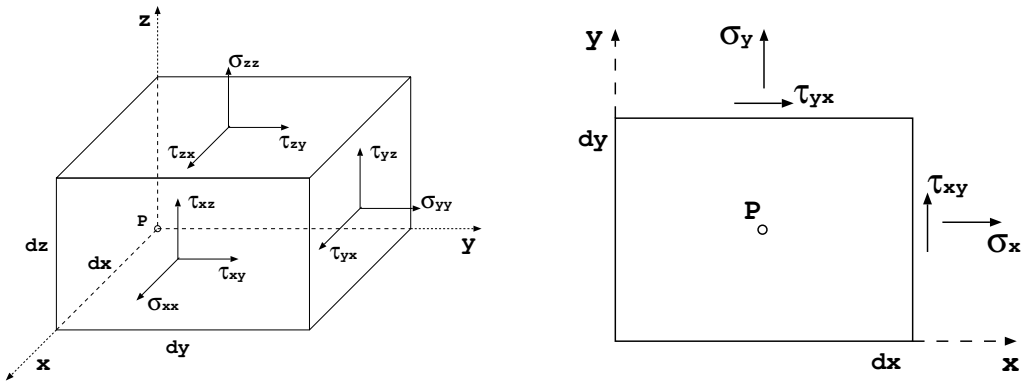
Dieser Grenzwert wird Spannungsvektor $\vec{t}$ genannt. Darüber ist der Spannungszustand in P allerdings noch nicht ausreichend beschrieben, da er abhängig von der Schnittrichtung ist. Zur genaueren Beschreibung wird in der Umgebung von P ein infinitesimaler Quader mit den Kantenlängen dx, dy und dz ausgeschnitten (Abb. 2.1 (a)). In jeder der sechs Flächen wirkt ein Spannungsvektor, der

zerlegt werden kann in eine Komponente σ normal und eine Komponente τ tangential zur Schnittfläche. Dies gilt auch entsprechend im Zweidimensionalen für ein Flächenelement mit den Kantenlängen dx und dy an den vier Kanten (Abb. 2.1 (b)). Die Tangentialkomponente τ wird Schubspannung genannt. Diese wird noch in die Komponenten nach den Koordinatenrichtungen zerlegt. Zur Kennzeichnung werden Doppelindizes verwendet. Dabei gibt der erste Index die Richtung der Flächennormale und der zweite die Richtung der Spannungskomponente an. Bei den Normalspannungen haben Flächennormale und Spannung die gleiche Richtung. Daher kann die Bezeichnung für σ vereinfacht werden durch

$$\sigma_{xx} = \sigma_x, \quad \sigma_{yy} = \sigma_y, \quad \sigma_{zz} = \sigma_z. \tag{2.24}$$

Es lässt sich zeigen, dass Schubspannungen in zwei senkrecht aufeinander stehenden Richtungen gleich sind, so dass gilt

$$\tau_{xy} = \tau_{yx}, \quad \tau_{xz} = \tau_{zx}, \quad \tau_{yz} = \tau_{zy}. \tag{2.25}$$



(a) Dreidimensional (b) Zweidimensional

Abbildung 2.1: Spannungszustand in der Umgebung eines Punktes P [2]

In der FEM können durch die berechneten Knotenverschiebungen (Kap. 2.1.1) die Dehnungen und mit dem Stoffgesetz (Hooke'sches Gesetz) die Spannungen in diesen Punkten abgeleitet werden. Bei der Betrachtung des einfachen Stabmodells ist die Dehnung ε definiert durch den Quotienten aus der Verlängerung Δl und der ursprünglichen Stablänge l

$$\varepsilon = \frac{\Delta l}{l}. \tag{2.26}$$

Wie die Spannungen bestehen die Dehnungen im Mehrdimensionalen jedoch aus mehreren Komponenten, die z.B. im Dreidimensionalen bezeichnet werden mit

$$\varepsilon_x,\ \varepsilon_y,\ \varepsilon_z,\ \gamma_{xy},\ \gamma_{yz},\ \gamma_{zx}. \tag{2.27}$$

Um die Dehnungen in beliebigen Knoten eines FEM-Berechnungsnetzes bestimmen zu können, werden Grundlagen der Kontinuumsmechanik benötigt. In dieser Theorie wird eine Struktur als Kontinuum betrachtet, das aus Materialpunkten besteht. Diese werden durch ihre Ortsvektoren $\vec{x}$ repräsentiert. Durch Einführung eines Koordinatensystems lässt sich $\vec{x}$ in seine Einzelkomponenten x_i zerlegen. Die Bewegung eines materiellen Punktes $\vec{x}$ in der Zeit t wird durch die Verschiebung $\boldsymbol{u}(\vec{x}, t)$ beschrieben. Der Zusammenhang zwischen Verschiebungen und Dehnungen in jedem

Punkt lässt sich dann im linearen Fall in symbolischer Tensor- bzw. in Komponentenschreibweise darstellen durch

$$\boldsymbol{\varepsilon} = \frac{1}{2}(grad\ \boldsymbol{u} + (grad\ \boldsymbol{u})^T) \qquad \Leftrightarrow \qquad \varepsilon_{ik} = \frac{1}{2}(\frac{\partial u_i}{\partial x_k} + \frac{\partial u_k}{\partial x_i}). \tag{2.28}$$

Über das Hooke'sche Gesetz lässt sich dann der Spannungstensor $\boldsymbol{\sigma}$ berechnen durch

$$\boldsymbol{\sigma} = \boldsymbol{E} : \varepsilon, \tag{2.29}$$

wobei $\boldsymbol{E}$ der Elastizitätstensor des verwendeten Materials ist.

Im Allgemeinen unterliegt ein Objekt einem mehrachsigen Spannungszustand. Um eine Aussage darüber machen zu können, wie stark die Beanspruchung des Objekts ist, muss eine geeignete Vergleichsspannung eingeführt werden, die alle Spannungskomponenten zu einer einzigen Spannung zusammenfasst. Dazu wird hier die **von Mises Spannung** σ_{vm} genutzt, die im ebenen Spannungszustand definiert ist durch

$$\sigma_{vm} = \sqrt{(\sigma_x + \sigma_y)^2 - 3(\sigma_x\sigma_y - \tau_{xy}^2)}. \tag{2.30}$$

Im Dreidimensionalen würde dies zu

$$\sigma_{vm} = \frac{1}{\sqrt{2}}\sqrt{(\sigma_x - \sigma_y)^2 + (\sigma_y - \sigma_z)^2 + (\sigma_z - \sigma_x)^2 + 6(\tau_{xy}^2 + \tau_{yz}^2 + \tau_{zx}^2)} \tag{2.31}$$

führen.

2.1.4 Adaptive Netzverfeinerung

Üblicherweise bleibt die Wahl eines für das jeweilige Problem passenden Berechnungsnetzes der Erfahrung des Ingenieurs überlassen. Es können jedoch auch nachträglich entsprechende Verfeinerungen vorgenommen werden, um die Ergebnisse der FEM-Berechnung zu verbessern. Dazu gibt es verschiedene Ansätze:

(i) p-Verfeinerung: Wahl von Elementen mit höherem Ansatz, d.h. Erhöhung der Ordnung der Polynome der Formfunktionen und damit der Anzahl an Knoten je Element (polynomial).

(ii) h-Verfeinerung: Reduzierung der Elementgröße, d.h. Erhöhung der Anzahl an Elementen (hierarchisch).

(iii) r-Verfeinerung: Umordnung der Knoten in dem Netz („re-arrange").

(iv) Kombinationen (h-p, r-p, r-h-p)

Die p-Verfeinerung wird dadurch realisiert, dass viele FEM-Programme über eine ganze Bibliothek von Elementen verfügen. So kann der Anwender den Elementtyp mit dem für das jeweilige Problem passenden Ansatz wählen. Die Struktur muss dann in deutlich weniger Elemente aufgeteilt werden, um Ergebnisse guter Genauigkeit zu erzielen. Elemente mit höherem Ansatz verursachen einen größeren Rechenaufwand, als einfache Elemente, da diese aus einer höheren Anzahl an Knoten bestehen. Dadurch müssen mehr Freiheitsgrade berechnet werden, was bedeutet, dass das Gleichungssystem sehr groß wird und viele Unbekannte bestimmt werden müssen. Bei der Betrachtung des Rechenaufwandes des Gesamtproblems ergibt sich aber eine Einsparung, da weniger Elemente nötig sind, als bei der h-Verfeinerung. Die minimale Ansatzgröße ergibt sich aus den wesentlichen inneren Randbedingungen, da diese durch die Formfunktionen erfüllt werden müssen.
Bei komplexen Strukturen und besonders, wenn Kerben im Objekt vorhanden sind, müssen aber trotzdem feine FEM-Netze erzeugt werden, um gute Ergebnisse zu erreichen. Die Genauigkeit der Berechnung lässt sich durch die Anzahl der finiten Elemente bei geeigneter Verfeinerung beliebig erhöhen. Um mit möglichst geringem Aufwand feine Netze erzeugen zu können, sind automatische

Netzgeneratoren erforderlich.
Gute Genauigkeiten für die Verschiebungen werden meist schon mit relativ wenigen Elementen erzielt. Gilt das Interesse aber anderen Größen, so muss dies keineswegs gelten. Dazu zählen auch die Spannungen, die notwendig sind, um Aussagen über die Sicherheit eines Systems bei Belastung machen zu können. Die Spannungen sind, genau wie andere physikalische Größen, in einigen Teilgebieten relativ groß, in anderen Gebieten jedoch kaum messbar. Gerade bei Strukturen mit scharfen Kerben treten im Bereich der Kerben sehr hohe Spannungen und Spannungsdifferenzen zwischen den Knoten auf. Deshalb sollte eine entsprechende Anpassung des Netzes an diese Gegebenheiten möglich sein. Für die Berechnung der Verformungen genügt normalerweise, je nach Anforderung an die Genauigkeit, ein eher grobes Netz. Um realistische Aussagen über Spannungen machen zu können, muss das Netz entsprechend verfeinert werden. Auch dazu gibt es automatische Programme wie FINEART (Kap. 4). Günstigerweise geschieht die Verfeinerung nur adaptiv an den empfindlichen Stellen, insbesondere an Kerben. Eine feinere homogene Vernetzung über die gesamte Struktur würde zu unnötig großen Elementzahlen und folglich auch Rechenaufwand führen. Um die kritischen Bereiche erkennen zu können, bedienen sich FEM-Programme eines „Fehlerschätzers“ (Kap. 2.1.5). Ändern sich die berechneten Spannungen von einem Punkt zu einem benachbarten Punkt stark, ist dies ein Maß für die Abweichung der berechneten von der wirklichen Spannung.
Die r-Verfeinerung bringt für die Genauigkeit der Ergebnisse keine Verbesserung. Sie ist sinnvoll für andere Optimierungen (Kap. 2.1.6).

2.1.5 Der FINEART-Fehlerschätzer

Wie in Kapitel 2.1.3 beschrieben, werden die Spannungen eines FEM-Netzes in einzelnen Punkten berechnet. Dies geschieht jedoch nicht in den Knoten des FEM-Netzes, sondern in den sogenannten **Gauß-Punkten**, die sich in der Umgebung der Knoten befinden, da diese die besten Stützstellen für die numerische Berechnung sind [13]. Dabei ergibt sich ein Fehler e_σ zwischen der berechneten Spannung σ und der tatsächlichen Spannung σ^* in den Gauß-Punkten

$$e_\sigma = \sigma^* - \sigma. \tag{2.32}$$

Da die exakten Spannungen σ^* nicht bekannt sind, müssen diese durch bessere Werte, als die berechneten ersetzt werden.
In FINEART werden dazu die Knotenpunktspannungen σ_{nd} durch Mittelung der Spannungen σ in Gauß-Punkten über die „superconvergent patch recovery technique“ [14] berechnet.

Da punktweise Fehler nur schwer zu berechnen sind, wird der Fehler in den einzelnen Elementen durch die Energienorm

$$\|e_e\| = \sqrt{\int_V e_\sigma^T \boldsymbol{E}^{-1} e_\sigma \, dV} \tag{2.33}$$

bestimmt. Der Gesamtfehler der Struktur kann dann durch Aufsummierung der Elementbeiträge berechnet werden.

$$\|e\|^2 = \sum_{e=1}^{n} \|e_e\|^2 \tag{2.34}$$

Der FINEART-Fehlerschätzer funktioniert nun folgendermaßen:

(i) Berechnung der Spannungen σ in den Gauß-Punkten

(ii) Berechnung von „verbesserten“ Spannungen σ_{nd} in den Knotenpunkten

(iii) Projezierung der Spannungen σ_{nd} über die Formfunktionen auf die Gauß-Punkte zur Berechnung der Spannungen σ^*_{nd}

(iv) Fehlerabschätzung der Struktur in der Energienorm

$$\|e\| = \sqrt{\sum_{V}^{n} \int (\sigma_{nd}^{*} - \sigma) \boldsymbol{E}^{-1} (\sigma_{nd}^{*} - \sigma) \, dV}. \tag{2.35}$$

(v) Abschätzung des relativen Fehlers η der Struktur durch

$$\eta[\%] = \frac{\|e\|}{\|u\|} \cdot 100, \tag{2.36}$$

wobei $\|u\|$ die exakte Energienorm ist;
und Abschätzung des relativen elementweisen Fehlers η_e durch

$$\eta_e[\%] = \frac{\|e_e\|}{\|e\|} \cdot 100. \tag{2.37}$$

Die verwendeten Fehlernormen wurden auch unabhängig von FINEART bereits genutzt [15].

2.1.6 Optimierung der Effizienz beim Lösen des Gleichungssystems

Die Finite-Elemente-Methode ist für die Berechnung sehr komplexer Modelle äußerst effektiv. Sie liefert in den meisten Fällen ein Ergebnis, wobei die Qualität der Lösung von der Güte des Berechnungsnetzes abhängig ist. Da sie bei komplexen Problemen oft zum Einsatz kommt, werden im folgenden Überlegungen angestellt, um die Effizienz der Finite-Elemente-Methode hierbei zu maximieren. Ziel ist dabei, die Rechnung in möglichst kurzer Zeit durchzuführen und möglichst wenig Speicherplatz in Anspruch zu nehmen.
Wie schon erwähnt wurde, besteht zumindest für statische, genügend komplexe Probleme ein großer Teil der Rechenarbeit bei der Finite-Elemente-Methode darin, lineare Gleichungssysteme zu lösen. Daher ist es sinnvoll die Zeit für diese Rechenarbeit zu minimieren, d.h. den besten Lösungsalgorithmus bzw. Alternativen zu finden. Im Allgemeinen werden spezielle Gleichungslöser für dünn besetzte Matrizen oder Bandmatrizen verwendet. Bei Löser für dünn besetzte Matrizen werden Algorithmen genutzt, die nicht an die Bandbreite gebunden sind. Hat die Matrix keine weiteren speziellen Eigenschaften, so wird für jeden Matrixeintrag ein Zeiger gespeichert, der die Position des Matrixelementes angibt. Durch den zusätzlich benötigten Speicher für die Zeiger führt diese Methode nur zu einer Verbesserung, falls die Anzahl der Matrixeinträge genügend klein ist.
Bei Bandmatrizen wird dieses Konzept nur verwendet, wenn die Bandbreite in den einzelnen Spalten (Zeilen) stark variiert oder die Bandstruktur selbst viele Nullelemente enthält. Bei variierender Bandbreite wird das Verfahren so erweitert, dass für jede Spalte (Zeile) ein Zeiger genutzt wird, der angibt wo die Nichtnullelemente beginnen. Die führenden Nullen werden so weggeschnitten, die inneren dagegen nicht. Befinden sich auch innerhalb der Bandstruktur viele Nullelemente wird entweder das Verfahren für allgemeine, dünn besetzte Matrizen genutzt oder die Matrix wird entsprechend umgeordnet, so dass die Anzahl der Nullelemente verringert wird. Dies ist z.B. über die r-Verfeinerung möglich, bei der die Knoten des Netzes entsprechend umgeordnet werden, so dass sich die Bandbreite der Steifigkeitsmatrix verkleinert. Dadurch wird auch die Rechenzeit bei Löser für Bandmatrizen optimiert.
Eine andere bzw. zusätzliche Möglichkeit die Rechenzeit zu minimieren, ist einen parallelen Löser einzusetzen. Dadurch wird das FEM-Problem auf mehrere Rechner verteilt, so dass jeder nur einen Teil des gesamten Problems zu bewältigen hat.

2.2 Einsatz von Parallelrechnern

Parallelrechner sind Computer mit mehreren Prozessoren [16] [17]. Die Prozessoren können gemeinsam an einer oder mehreren Aufgaben arbeiten. Die Leistungsfähigkeit eines Parallelrechners

wird aus Hardware orientierter Sicht durch die Rechengeschwindigkeit der Einzelprozessoren, die Übertragungsraten, sowie die Struktur des Verbindungsnetzwerks und die Organisation des Speichers bestimmt. Auf der Software-Ebene nimmt das verwendete Programmiermodell, das in der Regel von der Hardware abhängig ist sowie die Programmiersprache Einfluss auf die Leistungsfähigkeit. Vorteile eines Parallelrechners ist die meist höhere Rechengeschwindigkeit und der größere, verfügbare Speicherplatz. Jedoch ist für eine effiziente Nutzung ein zusätzlicher Programmieraufwand nötig.

2.2.1 Architekturen

Beispielsweise durch die Vernetzung mehrerer Computer über ein LAN (Local Area Network) oder auch WAN (Wide Area Network) kann ein Parallelrechner konstruiert werden. Dadurch werden die Vorteile eines Parallelrechners verdeutlicht: In diesem System besteht die Gesamt-Rechenleistung, sowie der vorhandene Speicherplatz aus der Summe der Ressourcen der einzelnen Computer. Dieser Parallelrechner ist jedoch für die meisten Probleme äußerst ineffizient, da das Verbindungsnetzwerk meist nur einen relativ langsamen Datenaustausch gewährleistet. Dies kann durch ein schnelleres Netzwerk verbessert werden. Solche Verbände, bestehend aus Einzel- oder Mehrprozessorsystemen, die durch ein Hochleistungsnetzwerk verbunden sind, werden als **Cluster** bezeichnet.
Weitere Optimierungen können eventuell durch Umstrukturierung der Speicherorganisation erzielt werden, so dass unterschiedliche physikalische Prozessortopologien entstehen. Zur genaueren Erläuterung wird zunächst die Einteilung von Parallelrechnern in Kategorien anhand der Organisation des Speichers beschrieben.
Die zwei bekanntesten Vertreter sind „shared memory“ Systeme und „distributed memory“ Systeme.

Bei einem „shared memory“ System (Abb. 2.2) haben allen Prozessoren einen gemeinsamen Speicherbereich. Die Transportwege der Daten werden von einem Verbindungsnetzwerk realisiert. Die Prozessoren kommunizieren mit Hilfe von gemeinsam benutzten Adressbereichen. Nachteile dieser Architektur liegen in der steigenden Komplexität bei einer Erhöhung der Prozessorenanzahl. Das Verbindungsnetzwerk wird dann zu aufwändig, denn der Speicher muss weiterhin für alle Prozessoren erreichbar sein. Zusätzlich ist es notwendig, die Speicherzugriffe mit Hilfe von Sperrmechanismen zu synchronisieren, um nicht zum Beispiel bei gleichzeitigen Schreibzugriffen einen inkonsistenten Speicherinhalt zu erhalten.
Haben alle Prozessoren die gleichen Zugriffsgeschwindigkeiten auf den gemeinsamen Speicher, so wird dies als **SMP**-System (symmetric multiprocessors, shared memory multiprocessors) bezeichnet. Das Verbindungsnetzwerk ist dann meist ein Bussystem oder ein Switch-basiertes System.

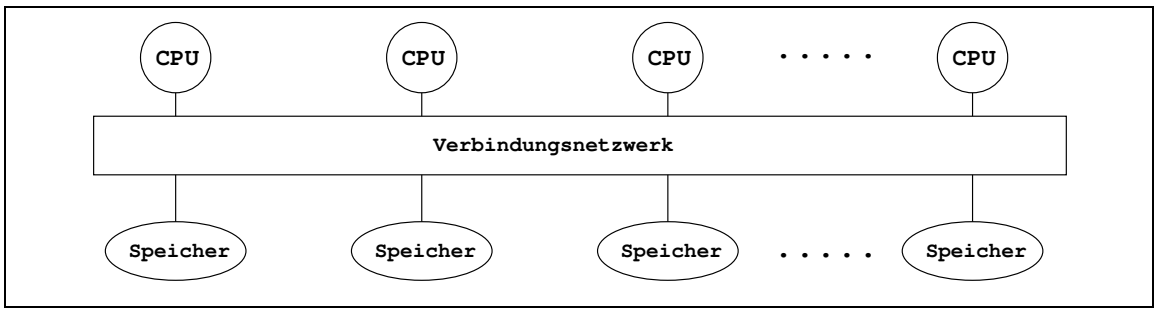


Abbildung 2.2: „shared memory“ System [16]

Bei „distributed memory“ Systemen (Abb. 2.3), also Systemen mit verteiltem Speicher, wird den Prozessoren je ein Speicherbereich physikalisch zugeordnet und jeder Prozessor hat im Allgemeinen nur auf seinen eigenen Speicher Zugriff. Der Datenaustausch zwischen den einzelnen Knoten, bestehend aus Prozessor und Speicher, kann in diesem Fall nicht mehr über gemeinsam benutzte

Adressbereiche erfolgen. Stattdessen werden Nachrichten verwendet, die vom Sender über das Verbindungsnetzwerk zum Empfänger gelangen.

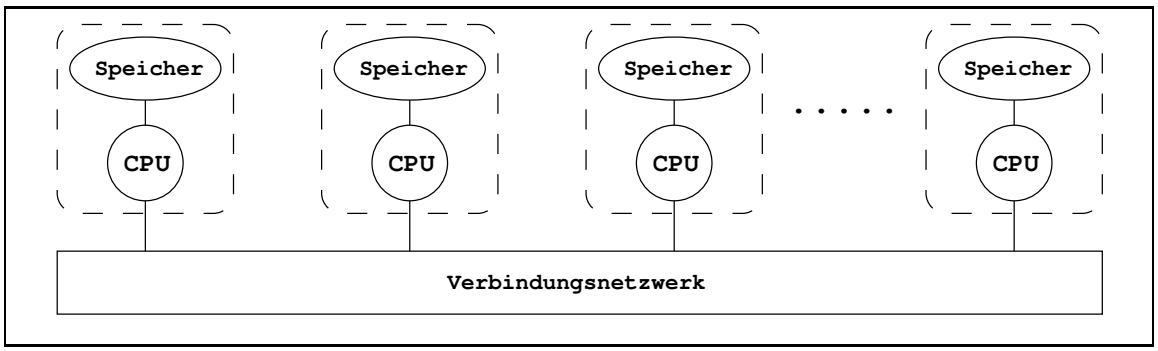


Abbildung 2.3: „distributed memory“ System [16]

In dieser Arbeit werden Untersuchungen (Kap. 3.5, 4.2) auf dem SMP-Cluster [18] ZAMpano (ZAM parallel nodes) des Zentralinstituts für Angewandte Mathematik (ZAM), Forschungszentrum Jülich GmbH durchgeführt (Abb. 2.4) [19].
Das ZAMpano besteht aus acht SMP-Knoten, die sich aus vier Prozessoren zusammensetzen. Die Knoten haben jeweils einen eigenen Speicherbereich und bilden ein „distributed memory“ System. Die Prozessoren eines Knotens greifen aber auf denselben Speicherbereich zu und sind somit ein „shared memory“ System.
Das ZAMpano ist eine Kombination beider Architekturen, was als **Hybridsystem** bezeichnet wird. Als Verbindungsnetzwerk wird „Myrinet“ eingesetzt, das alle SMP-Knoten direkt über einen Switch miteinander verbindet. Es gehört zur Gruppe der SANs (System Area Network), bei denen die Kommunikation im Vergleich zu LANs ohne Umweg über das Betriebssystem abläuft.

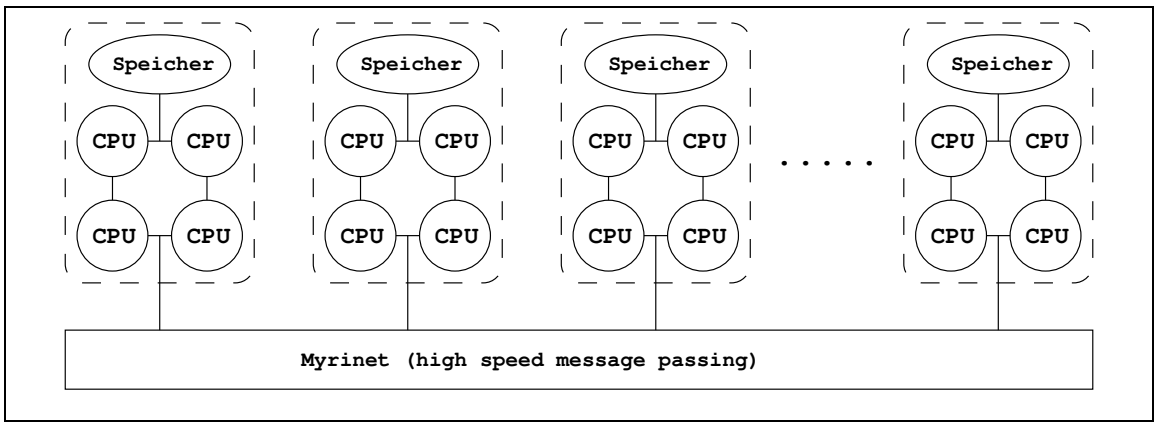


Abbildung 2.4: Architektur des ZAMpano [19]

Werden Parallelrechner mit einer sehr großen Zahl an Prozessoren genutzt, ist es kaum noch möglich eine direkte Verbindung aller Prozessoren zu verwalten. Bei der CRAY T3E [20], die aus 512 Prozessoren besteht, ist das Netzwerk durch einen 3D-Torus realisiert. Abbildung 2.5 zeigt ein Gitter, das entsprechend aufgebaut ist, allerdings nur zweidimensional. Das Netzwerk ist wesentlich schneller, dafür aber auch kostspieliger. Die Cray ist ein MPP-System (massive parallel processors) mit physikalisch verteiltem Speicher, der logisch auch global adressierbar ist. Diese Maschine wird im Rahmen dieser Arbeit neben dem ZAMpano ergänzend für Messungen benutzt, um Hardware abhängige Vergleichswerte für die Rechenzeiten zu erhalten.

Abbildung 2.6 zeigt die Entwicklung der schnellsten Computer der Welt [21]. Dabei wird die Anzahl an Computern jeder Architektur in der Liste der 500 schnellsten Maschinen als Funktion der Zeit angegeben. Einprozessormaschinen sind dort seit 1997 nicht mehr vertreten. Auch die SMPs

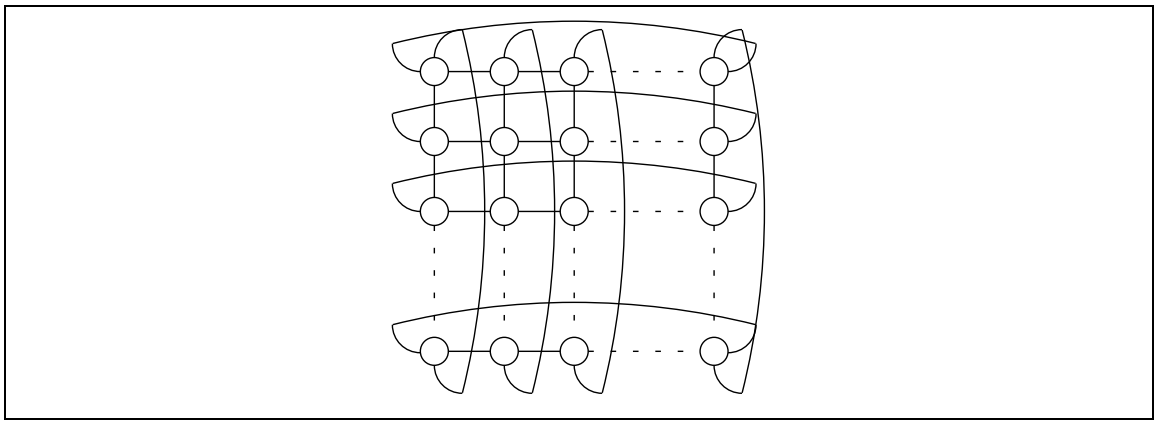


Abbildung 2.5: Architektur der CRAY T3E (vereinfacht)

scheinen aus dieser Liste herauszufallen, da mit einer steigenden Anzahl von Prozessoren der Zugriff auf einen gemeinsamen Speicher nur schwer realisiert werden kann. Dieser Nachteil wird von den MPPs, die einen verteilten Speicher besitzen, vermieden. MPPs stellen auch die meisten Rechner in der Top 500 Liste. In letzter Zeit sind jedoch immer mehr Cluster in dieser Liste vertreten. Durch diese Technik ist es möglich, auf günstige Weise mittels SMPs eine Variation von MPPs zu realisieren. Aus diesem Grund werden verstärkt Analysen dieser Systeme durchgeführt, wodurch immer leistungsfähigere Cluster entstehen.

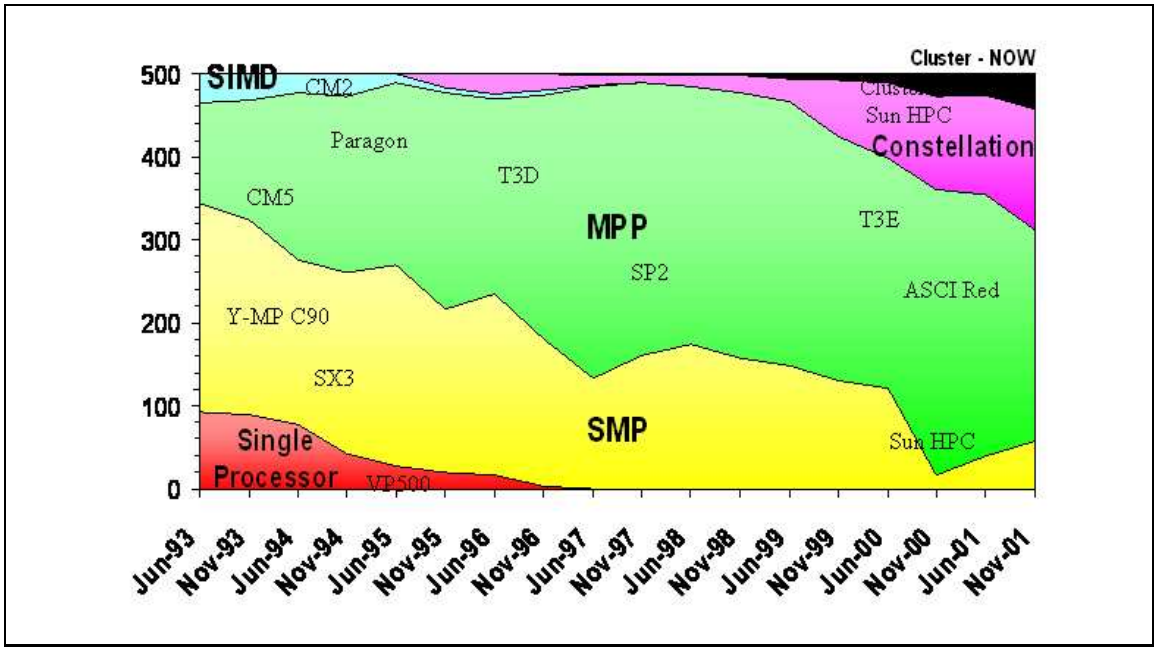


Abbildung 2.6: Entwicklung der Architekturen der 500 schnellsten Computer der Welt bis November 2001 [21]

2.2.2 Programmierparadigmen

Je nach der Art des Speicherzugriffs (gemeinsamer oder verteilter Speicher), muss die Programmiertechnik angepasst werden. Dazu werden nicht mehr die einzelnen Prozessoren, sondern die logischen **Prozesse** betrachtet, die auf ihnen ausgeführt werden. Wird ein Programm mit konkreten Daten abgearbeitet, wird dieser Vorgang Prozess genannt. Der Prozess beginnt seine Existenz mit der ersten konkret ablaufenden Aktion und hört mit seiner letzten Aktion auf zu existieren. Jede einzelne Abarbeitung eines Programms ist ein einmaliger Prozess.
Es ergeben sich neben den zuvor genannten physikalischen Prozessortopologien virtuelle Prozess-

topologien. Dies ist die vom Nutzer gewünschte oder vom Betriebssystem vorgegebene Vernetzung der Prozesse. Sie wird meist von der Daten- bzw. Kommunikationsstruktur bestimmt (Kap. 3.2).
Nach Einführung des Begriffes „Prozess“ sei an dieser Stelle bemerkt, dass bei Effizienzuntersuchungen von Parallelrechnern in Bezug auf die Rechenzeit zwischen den beiden folgenden Größen unterschieden werden kann. Die Erste ist die Wartezeit des Benutzers („wall-clock time“), die durch den Prozess bestimmt wird, der die meiste Rechenzeit benötigt. Dies ist die meist bedeutungsvollere beider Größen. Die zweite ist die CPU-Zeit, die sich durch aufsummieren der Einzelzeiten der Prozesse ergibt. Sie spielt dann eine Rolle, wenn Kenngrößen wie Ressourcenverbrauch untersucht werden sollen.

Eine Möglichkeit ein Problem parallel zu lösen, ist ein Programm parallel auf mehreren Prozessen gleichzeitig auszuführen, wobei jeder Prozess unterschiedliche Daten bearbeitet. Diese Methode wird als Datenparallelismus oder SPMD (Single Program Multiple Data) bezeichnet. Im entgegengesetzten Fall werden für ein Problem mehrere Programme geschrieben, wobei jedes ein Teilgebiet der Aufgabenstellung berechnet und die Programme parallel ausgeführt werden. Dieser Ansatz wird Aufgabenparallelismus oder MPMD (Multiple Program Multiple Data) genannt.

Bei der Programmierung eines „distributed memory“ Systems, geschieht die Kommunikation zwischen den Prozessen in der Regel über den Austausch von Nachrichten, während bei einem „shared memory“ System gemeinsame Adressbereiche bzw. Variablen verwendet werden. Es ist jedoch auch möglich, auf einem System mit gemeinsamen Speicher Nachrichten zu verwenden bzw. auf nachrichtenbasierten Systemen gemeinsame Variablen. Beide Techniken werden eingesetzt; bei der zweiten muss eine Abstraktionsschicht auf Ebene des Betriebssystems eingeführt werden, welche die Abbildung der gemeinsamen Variable auf die physikalisch getrennten Adressbereiche übernimmt (wie bei der CRAY T3E).

Bei der Kommunikation mit Hilfe von Nachrichten (Abb. 2.7) laufen zwei oder mehrere Prozesse parallel auf einer Architektur mit verteiltem Speicher nebeneinander ab. Dabei kann sowohl das SPMD-Modell, als auch das MPMD-Modell eingesetzt werden. Wesentlich häufiger wird jedoch das SPMD-Modell verwendet, bei dem von allen Prozessen das gleiche Programm repliziert bearbeitet wird. Die Prozesse von nachrichtenbasierten parallelen Programmen werden in verschiedenen Adressräumen ausgeführt. Variablen eines Prozesses können nicht von anderen Prozessen gelesen bzw. verändert werden. Der Programmierer ist für den gesamten Datenaustausch verantwortlich, was sowohl die Datenaufteilung, Kommunikation und Synchronisation als auch das Allozieren und Freigeben aller Speicherbereiche mit einschließt.
Seit 1997 hat sich zur Realisierung dieses Konzeptes das „Message Passing Interface“ MPI [22] als Standard etabliert, das mit allen gängigen Programmiersprachen genutzt werden kann. Daneben gibt es weitere Bibliotheken wie PVM („Parallel Virtual Machine“).

Die Kommunikation über gemeinsame Variablen entspricht dem SPMD-Modell, wobei die einzelnen Prozesse ihre Daten mit Hilfe eines gemeinsam benutzten Adressbereiches austauschen. Dies setzt voraus, dass die Architektur einen derartigen Bereich zur Verfügung stellt, auf dem alle Prozesse sowohl lesend, als auch schreibend zugreifen können. Eine Möglichkeit dieses Programmierparadigma zu verwenden, ist der Einsatz von „Threads“ (Abb. 2.8). Dies sind parallel ablaufende Steuerflüsse innerhalb eines Prozesses. Das Programm wird grundsätzlich von einem „Master-Thread“ abgearbeitet. Über entsprechende Direktiven werden dann parallele Regionen gekennzeichnet, die von mehreren „Threads“ parallel bearbeitet werden. Typischerweise sind dies Schleifen. Jeder erzeugte „Thread“ besitzt seinen eigenen Stapelspeicher und seinen eigenen Befehlszähler. Der gemeinsame Speicherbereich wird über den „Master-Thread“ zur Verfügung gestellt. Dieser Speicher wird auch zur Synchronisation und zum Datenaustausch verwendet. Um eine korrekte Ausführung

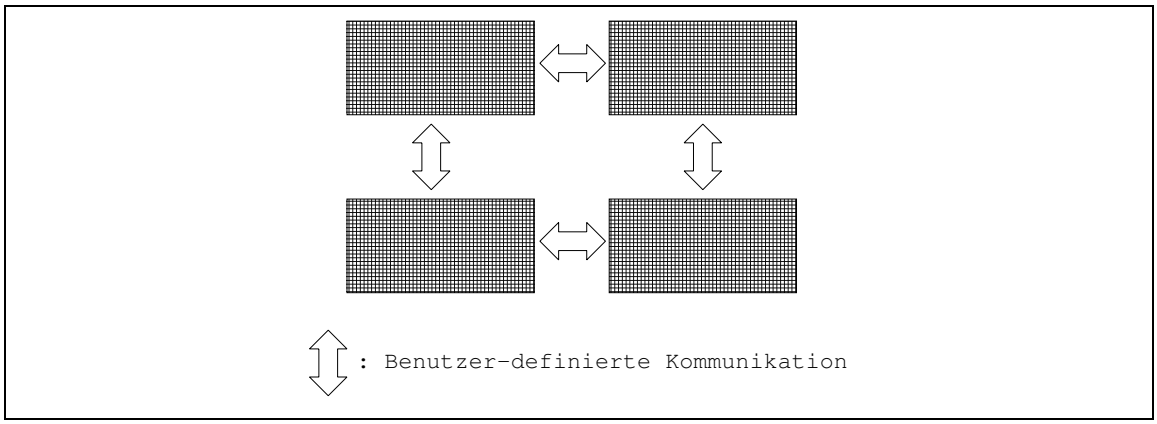


Abbildung 2.7: Kommunikation über Nachrichten

aller Prozesse zu gewährleisten, müssen die Speicherzugriffe auf gemeinsame Variablen synchronisiert werden.
Auch hier hat sich in den letzten Jahren ein Standard entwickelt. Seit 1997 existiert für FORTRAN 77 OpenMP, ein Satz von Direktiven und Laufzeitroutinen und seit 1998 ein entsprechender Satz von Pragmas für C und C++. Die Arbeiten an einer Erweiterung von OpenMP für FORTRAN 95 sind noch nicht beendet.

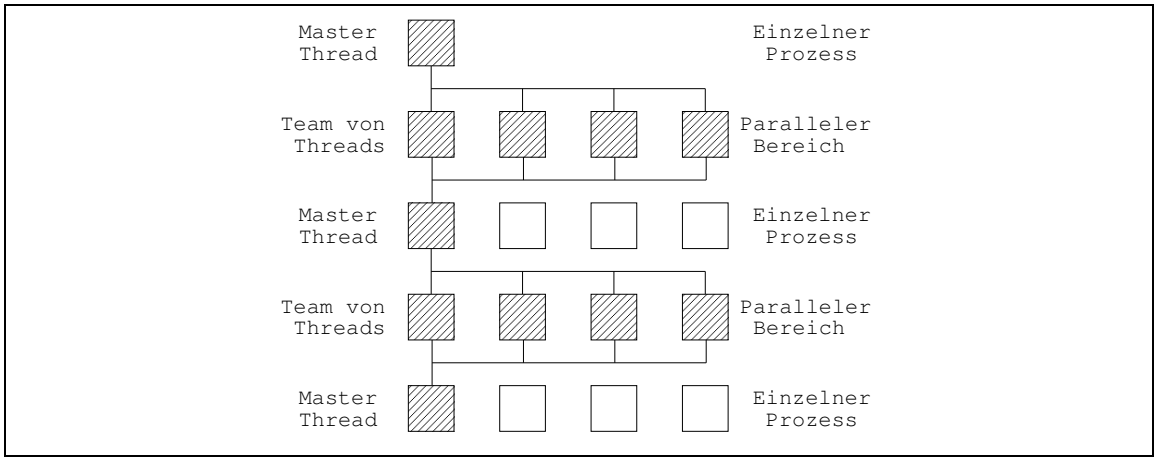


Abbildung 2.8: Kommunikation über gemeinsam benutzte Adressbereiche

Der Vollständigkeit halber sei noch das reine SPMD-Modell der Datenparallelität erwähnt, bei dem parallele Operationen synchron auf allen Elementen großer Datenmenge ausgeführt wird. Dies sind typischerweise Matrizen und Vektoren. Bei der Programmierung wird eine sequenzielle Semantik genutzt. Seit 1996 ist HPF (High Performance FORTRAN) Standard.

Es stellt sich nun die Frage, welches Programmiermodell für das ZAMpano am besten geeignet ist. Betrachtet man einen SMP-Knoten des ZAMpano, ist dies ein „shared memory“ System. Das bedeutet für die Nutzung nur eines Knotens wäre eine Programmentwicklung über gemeinsame Adressbereiche sinnvoll. Wird andererseits die Struktur der SMP-Knoten selbst vernachlässigt, so ist das ZAMpano aus globaler Sicht ein „distributed memory“ System, so dass die Nachrichtenkommunikation bei der Programmierung zu bevorzugen ist. Letztendlich wäre ein hybrides Programmiermodell das effizienteste, jedoch auch aufwändigste. Zudem gibt es kaum vorhandene problemspezifische Programmbibliotheken, die dies unterstützen. Um trotzdem alle Ressourcen sinnvoll nutzen zu können, wird in dieser Arbeit die Kommunikation mit Hilfe von Nachrichten durchgeführt.

2.2.3 Kenndaten der verwendeten Parallelrechner

In Tabelle 2.1 sind die wichtigsten Kenndaten des ZAMpano und der CRAY T3E zum Vergleich angegeben. Wie bereits erwähnt, ist die Gesamt-Prozessorzahl der CRAY T3E wesentlich höher, als die des ZAMpano. Dabei stehen auf dem ZAMpano insgesamt 32 Prozessoren auf 8 SMP-Knoten für Parallelrechnungen zur Verfügung. Zusätzlich existiert ein Service-Knoten, ebenfalls bestehend aus 4 Prozessoren. Die einzelnen Prozessoren der beiden Systeme unterscheiden sich nur geringfügig in der Taktfrequenz. Da es sich jedoch um verschiedene Prozessortypen handelt, ist dies kein aussagekräftiges Vergleichmaß für die Rechenleistung. Stattdessen wird die Anzahl an Gleitkomma-Operationen je Sekunde (FLOPS) miteinander verglichen. Diese ist bei einem Prozessor der CRAY T3E etwa doppelt so hoch, wie auf dem ZAMpano. Wegen der wesentlich höheren Prozessorzahl beträgt die Gesamtleistung der CRAY T3E etwa das 30-fache des ZAMpano.
Der verfügbare Arbeitsspeicher je Knoten auf dem ZAMpano entspricht dem 4-fachen der CRAY T3E. Da allerdings ein Knoten auf der CRAY T3E nur aus einem Prozessor besteht, auf dem ZAMpano jedoch aus vier, ist der Arbeitsspeicher je Prozessor anteilsmäßig identisch. Die zu verarbeitenden Daten müssen jedoch eventuell, wie auch in dieser Arbeit, zunächst von einem Hauptprozess eingelesen werden. Diese Zwischenlagerung könnte einen Engpass darstellen, da der zusätzliche Speicher der übrigen Prozessoren auf der T3E nicht zur Verfügung steht. Bei dieser Einleseprozedur erweist sich dann der vierfache, zur Verfügung stehende Speicherplatz des ZAMpano für einen Knoten bei großen Datenmengen von Vorteil. Dies lässt sich allerdings umgehen, indem die Daten vor Beendigung des Einlesens bereits teilweise an die übrigen Prozesse verteilt werden. Falls für die parallele Bearbeitung einer Aufgabe jedoch weniger als drei Prozesse genutzt werden, stellt sich erneut das Problem, dass dann sogar während der kompletten Aufgabenbearbeitung auf der T3E weniger Speicher je Prozess verfügbar ist. Im Allgemeinen ist dies nicht gegeben, da bei Ausführung eines parallelen Programms mehr als drei Prozesse genutzt werden. Für Messungen (Kap. 3.5, 4.2) könnte das Problem jedoch von Bedeutung sein.
Die Tabelle macht ebenfalls deutlich, dass das Verbindungsnetzwerk der CRAY T3E wesentlich schneller ist, als das des ZAMpano. Die Latenzzeit gibt dabei die Verweildauer der Datenpakete in einem Netzwerkknoten an, bis sie bearbeitet und weitergeleitet worden sind. Sie ist, ebenso wie die Bandbreite, abhängig von der Nachrichtengröße, dem Kommunikationssystem und auf dem ZAMpano zusätzlich von den verwendeten Prozessoren (Kap. 3.5.1). Diese Werte gelten für MPI bei minimaler Größe der Nachrichtenpakete. Sowohl Latenzzeit, als auch Bandbreite sind auf der CRAY T3E etwa um das 10-fache besser als auf dem ZAMpano. Dieser enorm große Unterschied kommt jedoch, neben der weniger leistungsfähigen Hardware, durch die aktuelle Konfiguration der MPI-Version des ZAMpano zustande.

Kenngröße		ZAMpano	CRAY T3E
Prozessorzahl		$8 \cdot 4$ (+ $1 \cdot 4$)	512
Prozessortyp		Intel Pentium III Xeon	Typ 1200 (DEC Alpha 21164)
Taktfrequenz		550 MHz	600 MHz
Leistung je Prozessor		550 MFLOPS	1200 MFLOPS
Gesamtleistung		19,8 GFLOPS	614 GFLOPS
Arbeitsspeicher je Knoten		2 GB	0,5 GB
Gesamtarbeitsspeicher		16 + 2 GB	262 GB
Latenzzeit	des Verbindungsnetzw.[1]	15-50 μs	3,75 μs
Bandbreite		22-36 MB/s	312 MB/s
Betriebssystem		SUSE Linux	UNICOS/mk

Tabelle 2.1: Kenndaten von ZAMpano und CRAY T3E

[1] bei minimaler Größe der Nachrichtenpakete mit MPI, Stand: Januar 2002

Kapitel 3

Lösen von Gleichungssystemen mit ScaLAPACK

Das skalierbare Lineare Algebra Paket ScaLAPACK ist eine parallele Programmbibliothek, die sowohl Routinen für die Berechnung linearer Gleichungssysteme, als auch für die Prozesskommunikation bereitstellt. Das bei der Finite-Elemente-Methode entstehende LGS kann mit Hilfe dieser Bibliothek parallel gelöst werden. Dazu muss die Steifigkeitsmatrix in Teilmatrizen, sowie die RHS in Teilvektoren zerlegt und diese auf die einzelnen Prozesse aufgeteilt werden. Die anschließend von ScaLAPACK berechneten Teillösungen müssen dann wieder zu einem Gesamtergebnis zusammengeführt werden.
ScaLAPACK bietet eine Reihe von direkten Lösern für dicht besetzte Matrizen und Bandmatrizen, jedoch keinen für dünn besetzte Matrizen. Bei den Steifigkeitsmatrizen handelt es sich um symmetrische, positiv definite Matrizen mit dünn besetzter Struktur, was bei entsprechender Knotennummerierung zu Bandmatrizen führt. In ScaLAPACK existiert ein Löser für diesen Typ von Bandmatrizen, jedoch keiner für dünn besetzte Matrizen.
Es wird sowohl dieser Löser als auch einer für dicht besetzte Matrizen untersucht. So können Vergleiche hinsichtlich der benötigten Rechenzeit und des Speicherplatzes getätigt werden. Die beiden Löser unterscheiden sich in der Art der Aufteilung der Teilmatrizen auf die einzelnen Prozesse und dem Lösungsalgorithmus.
Die nun folgenden Erläuterungen wurden der nachstehenden Referenz entnommen[23].

3.1 Funktion und Aufbau von ScaLAPACK

ScaLAPACK ist eine Bibliothek mit linearen Algebra Routinen für „distributed memory“ Systeme. Es berechnet lineare Gleichungssysteme $A\vec{x} = \vec{b}$, Probleme der kleinsten Fehlerquadrate $\min_x \|\vec{b} - A\vec{x}\|_2$ und Eigenwertprobleme. Um die Datenkommunikation zwischen den einzelnen Prozessen minimal zu halten, werden blockaufteilende Algorithmen verwendet (Kapitel 3.3).
ScaLAPACK ist eine Weiterentwicklung von LAPACK, das für „shared memory“ Systeme geschrieben wurde. LAPACK nutzt die bereits vorhandenen, optimierten BLAS (Basic Linear Algebra Subprograms), um herkömmliche lineare Algebra Berechnungen, wie Skalarprodukt, Matrix-Vektor Multiplikation und Matrix-Matrix Multiplikation durchführen zu können. Analog hierzu gibt es für ScaLAPACK das parallele Werkzeug PBLAS (Parallel BLAS)[24].
Um eine virtuelle Prozesstopologie erstellen und zwischen den einzelnen Prozessen kommunizieren zu können, existiert als weitere Komponente BLACS[25] (Basic Linear Algebra Communication Subprograms), das genau wie MPI eine „message passing“ Bibliothek ist (Kap. 3.2). Abbildung 3.1 illustriert die Hierarchie der einzelnen Programmpakete.

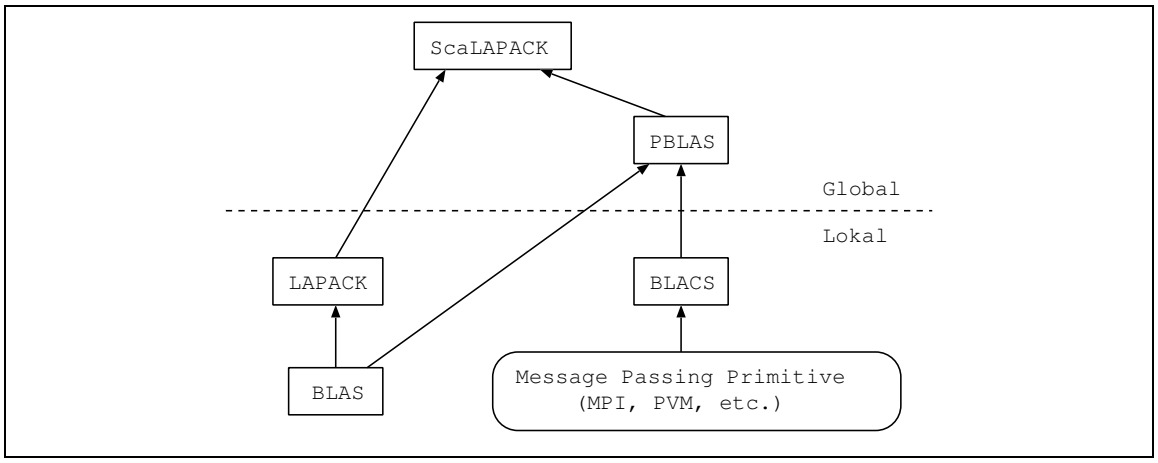


Abbildung 3.1: ScaLAPACK Software Hierarchie [23]

Dabei werden lokale Komponenten auf einem einzelnen Prozess ausgeführt. Globale Komponenten sind synchrone, parallele Routinen, deren Argumente Matrizen und Vektoren beinhalten, die auf mehreren Prozessoren verteilt sind.

3.2 Erstellung einer virtuellen Prozesstopologie mit BLACS

Die P Prozesse, mit denen ein Programm auf dem Parallelrechner gestartet wird, werden im Allgemeinen durch ein eindimensionales lineares Feld mit den Elementen $0, 1, ..., P-1$ dargestellt. Oft ist es aber praktisch, die Gesamtzahl der Prozesse P in ein zweidimensionales Gitter aufzuteilen, das P_r Prozessreihen und P_c Prozessspalten hat, so dass $P_r \cdot P_c = P$ ist (Abb. 3.2[2]). Dies gilt insbesondere für Lineare Algebra Probleme, bei denen mit Matrizen, sowie Vektoren und Skalaren, die Unterklassen der Matrizen sind, gerechnet wird. Aus diesem Grund basiert die Kommunikation von BLACS auf einem zweidimensionalen Konzept, anstatt auf einem eindimensionalen, wie es z.B. standardmäßig bei MPI der Fall ist. Dort muss zur Nutzung eines mehrdimensionalen Kommunikationskonzepts zunächst eine geeignete Prozesstopologie manuell initialisiert werden, die beliebig dimensioniert sein kann. Mit BLACS hingegen kann die Kommunikationsstruktur auf ein eindimensionales Feld reduziert werden, indem ein Parameter zur Bestimmung des Prozessgitters mit eins gewählt wird.
Bei der Implementierung eines ScaLAPACK Gleichungslösers in ein Programm ist die Dimensionierung des Prozessgitters sowie das Verhältnis zwischen P_c und P_r von dem verwendeten Lösungsalgorithmus abhängig. Dieser bestimmt die Datenaufteilung und die notwendigen Kommunikationsvorgänge. Je nach Kommunikationsfluss kann so die Effizienz des Algorithmus gesteigert werden [26]. Wie später noch belegt wird, ist es sinnvoll, bei Bandmatrizen ein eindimensionales Feld und bei dicht besetzten Matrizen ein zweidimensionales Gitter zu verwenden.
Bei der Implementierung des Lösers für symmetrische, positiv definite, dicht besetzte Matrizen ist die Datenkommunikation gleichmäßig in Richtung der Zeilen und Spalten (Kapitel 3.4.2). Daher sollte das Prozessgitter hier möglichst quadratisch sein, d.h. die Dimensionierungsparameter sollten so gewählt werden, dass P_r möglichst nah an P_c liegt. Bei anderen Lösern wird bei jedem Schritt des Algorithmus ein Block von Zeilen bzw. Spalten an die anderen Prozesse gesendet. Es ist möglich, diese Kommunikationsphase so durchzuführen, dass es zu Überschneidungen mit einigen Rechnungen kommt und somit die Anzahl der Kommunikationsprozesse in der Regel verringert wird. Die Richtung der Kommunikation bestimmt die Form des Gitters.
Das Gitter kann sowohl spalten- als auch zeilenweise angelegt werden.

[2] In der Abbildung werden, wie auch im Folgenden, die Prozesskoordinaten mit dargestellt

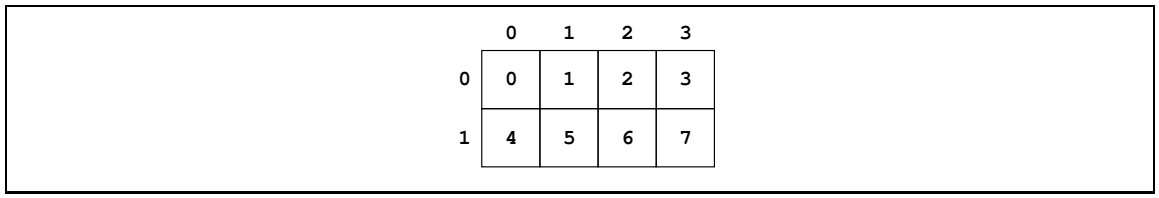


Abbildung 3.2: Acht Prozesse zeilenweise aufgeteilt in ein 2×4 Prozess-Gitter [23]

3.3 Aufteilung der Daten auf die verschiedenen Prozesse

Die Art, wie die Daten aufgeteilt werden, beeinflusst die Effizienz der Algorithmen stark. Daher ist es sinnvoll, zunächst ein möglichst geeignetes Schema zu bestimmen.
Wichtig ist, dass einerseits die Datenlast gut balanciert ist, d.h. dass jeder Prozess etwa die gleiche Datenmenge verarbeitet. Andererseits sollte die Datenkommunikation zwischen den einzelnen Prozessen minimal gehalten werden. Je nachdem, welcher Algorithmus genutzt wird, gibt es verschiedene Schemata, die diese beiden Faktoren optimieren. Die folgenden Schemata werden von ScaLAPACK für dicht besetzte bzw. für Bandmatrizen benutzt. Die Wahl des Schemas richtet sich nach den Eigenschaften bzw. dem Datenfluss bei der Berechnungsmethode des jeweiligen Algorithmus.

3.3.1 Eindimensionale Spalten- bzw. Zeilenblock Aufteilung

Bei der Spaltenblock-Aufteilung werden die Matrixspalten in zusammenhängende Blöcke aufgeteilt, deren Anzahl genau der Anzahl der Prozesse entspricht. Jeder Prozess erhält genau einen Spaltenblock. Daraus ergibt sich, dass Spalte k auf Prozess k/nb gespeichert wird, wobei die Spalten-Blockgröße

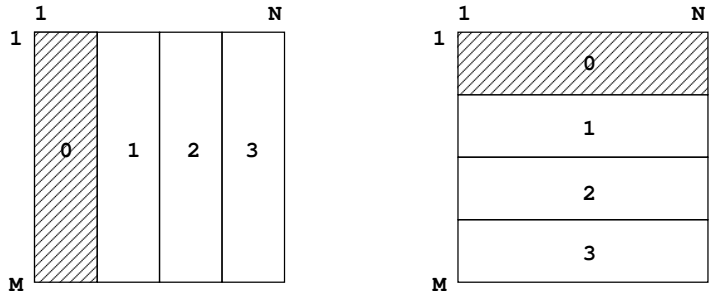
$$nb = \frac{\textit{Anzahl Spalten}}{\textit{Anzahl Prozesse}} \tag{3.1}$$

die maximale Anzahl an gespeicherten Spalten je Prozess ist. Ist diese Division nicht ohne Rest durchführbar, wird nb entsprechend vergrößert, um auch die restlichen Spalten aufteilen zu können. Der letzte Prozess erhält dann einen kleineren Block.
Dieses Schema wird für **Bandmatrizen** genutzt, da es zu einer effizienten Implementierung des von ScaLAPACK verwendeten „Divide and Conquer“ Algorithmus führt (Kap. 3.4.3). Bei dieser Aufteilung wird vorausgesetzt, dass die Blockgröße nicht kleiner wird, als $2 \cdot \beta + 1$, damit der Lösungsalgorithmus funktioniert, wobei β die Bandbreite der Matrix ist. Dies hat zur Folge, dass bei großen Bandbreiten nur eine begrenzte Anzahl an Prozessen genutzt werden kann.
Die Zeilenblock-Aufteilung läuft analog ab, mit dem Unterschied, dass dabei die Zeilen in Blöcke zerlegt werden, was für die RHS von Bandmatrizen benutzt wird, da diese meist nur eine Spalte enthält. Es wird dann entsprechend die Zeilen-Blockgröße

$$mb = \frac{\textit{Anzahl Zeilen}}{\textit{Anzahl Prozesse}} \tag{3.2}$$

verwendet.
Abbildung 3.3 veranschaulicht dieses Prinzip, wobei die Prozesse von 0 bis $P-1$, die Matrixzeilen von 1 bis M und die Matrixspalten von 1 bis N nummeriert werden.

Da bei Bandmatrizen viele Nullelemente existieren, wird dieses Schema so erweitert, dass möglichst wenige dieser Elemente ohne Informationsgehalt gespeichert werden. Dies geschieht, indem man die Elemente der Spalten so verschiebt, dass die unteren Zeilen der Matrix Nullzeilen werden, die dann nicht mehr gespeichert werden müssen (Abb. 3.4):
Die Spalte, in der das letzte Element der ersten Zeile ungleich Null ist, wird nicht verschoben (Spalte 3 in Abb. 3.4). Alle Elemente rechts davon werden spaltenweise zusammenhängend nach oben



(a) Zeilenblock Aufteilung

(b) Spaltenblock Aufteilung

Abbildung 3.3: Eindimensionale Aufteilungen bei 4 Prozessen [23]

verschoben. Das erste Element jeder Spalte innerhalb der Bandstruktur wird dann in die erste Zeile eingetragen. Alle Elemente links von der unveränderten Spalte werden entsprechend nach unten verschoben. Die Zeilenanzahl der umgeordneten Matrix entspricht $2 \cdot \beta + 1$; die Spaltenanzahl bleibt unverändert. Die neue Matrix ist demnach im Gegensatz zur ursprünglichen nicht mehr quadratisch.

$$\text{A=}\begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & 0 & 0 \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & 0 \\ 0 & 0 & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ 0 & 0 & 0 & a_{64} & a_{65} & a_{66} & a_{67} \\ 0 & 0 & 0 & 0 & a_{75} & a_{76} & a_{77} \end{pmatrix} \rightarrow$$

0			1			2
$*$	$*$	a_{13}	a_{24}	a_{35}	a_{46}	a_{57}
$*$	a_{12}	a_{23}	a_{34}	a_{45}	a_{56}	a_{67}
a_{11}	a_{22}	a_{33}	a_{44}	a_{55}	a_{66}	a_{77}
a_{21}	a_{32}	a_{43}	a_{54}	a_{65}	a_{76}	$*$
a_{31}	a_{42}	a_{53}	a_{64}	a_{75}	$*$	$*$

Abbildung 3.4: Speicherung einer nichtsymmetrischen Bandmatrix A [23]

Wie deutlich erkennbar ist, hat sich die Anzahl der Elemente ohne Informationsgehalt, die gespeichert werden, verringert. Nullen innerhalb der Bandstruktur werden dabei nicht berücksichtigt. Existiert dort ein Eintrag am Rand der Matrix, vergrößert dieser Ausreißer die komplette Bandstruktur und somit die Zeilenanzahl der umgeordneten Matrix.

Bei **symmetrischen Bandmatrizen** kann die Effizienz der Speicherung noch weiter gesteigert werden, indem nur entweder die untere oder die obere Dreiecksmatrix von A berücksichtigt wird. Abbildung 3.5 zeigt dies für den Fall, dass die untere Dreiecksmatrix gespeichert wird. Der Unterschied zur Speicherung bei nichtsymmetrischen Bandmatrizen ist, dass nur noch die $\beta + 1$ unteren (oberen) Zeilen gespeichert werden.

0			1			2
a_{11}	a_{22}	a_{33}	a_{44}	a_{55}	a_{66}	a_{77}
a_{21}	a_{32}	a_{43}	a_{54}	a_{65}	a_{76}	$*$
a_{31}	a_{42}	a_{53}	a_{64}	a_{75}	$*$	$*$

Abbildung 3.5: Speicherung der unteren Dreiecksmatrix einer symmetrischen Bandmatrix A [23]

3.3.2 Zweidimensionale blockzyklische Aufteilung

Zur Veranschaulichung wird zunächst einmal nur die eindimensionale spaltenblockzyklische Aufteilung betrachtet. Wichtig ist, dass bei blockzyklischen Aufteilungen eine feste Blockgröße nb für die Spaltenaufteilung bzw. mb für die Zeilenaufteilung gewählt wird. Wie später noch an Beispielen belegt wird, nimmt gerade diese Blockgröße Einfluss auf die Effizienz des Algorithmus [27]. Bei einer kleinen Blockgröße können die Daten gleichmäßiger aufgeteilt werden, eine große dagegen verringert die notwendige Datenkommunikation. Da die optimale Blockgröße auch von anderen Faktoren, wie Rechnerarchitektur, Prozesszahl, Dimensionierung des Prozessgitters, usw. abhängig ist (Kap. 3.5.2), ist es schwierig eine allgemeine Formel für diese Blockgröße zu finden, was bisher auch nicht gelungen ist [26].
Die Matrixspalten werden wieder in zusammenhängende Blöcke zerlegt, wobei jetzt im Normalfall je nach gewählter Blockgröße einem Prozess mehrere Blöcke zyklisch zugewiesen werden. Anschaulich bedeutet das, dass Block 0 Prozess 0 zugeordnet wird, ..., Block $(P_r - 1)$ Prozess $(P_r - 1)$, Block P_r wieder Prozess 0, usw. (Abb. 3.6 (a)).

Im Zweidimensionalen wird das selbe Prinzip angewandt. Es werden aber sowohl Spalten als auch Zeilen in Blöcke zerlegt und anschließend den jeweiligen Prozessen zugewiesen. Die Blöcke werden entsprechend der virtuellen Prozesstopologie zyklisch verteilt. Abbildung 3.6 (b) zeigt eine zweidimensionale blockzyklische Aufteilung auf einem 2×2 Prozess-Gitter, das zeilenweise angelegt wurde. So wird jede Block-Zeile der Matrix nach der eindimensionalen Methode entsprechend dem virtuellen Prozessgitter verteilt. In diesem Beispiel gilt für die erste Block-Zeile, dass Block 0 Prozess 0 zugeordnet wird, Block 1 Prozess 1, Block 2 wieder Prozess 0 usw. Für die zweite Block-Zeile gilt dann bei neuer Nummerierung der Blöcke, dass Block 0 Prozess 2 zugeordnet wird, Block 1 Prozess 3, Block 2 wieder Prozess 2 usw. Ab der dritten Zeile wiederholt sich dieses Procedere. Dieses Schema wird für **dicht besetzte Matrizen** benutzt. So kann in Abhängigkeit von der Blockgröße die Balance zwischen Datenlast und notwendiger Datenkommunikation sowohl in Richtung der Zeilen als auch der Spalten optimiert werden.

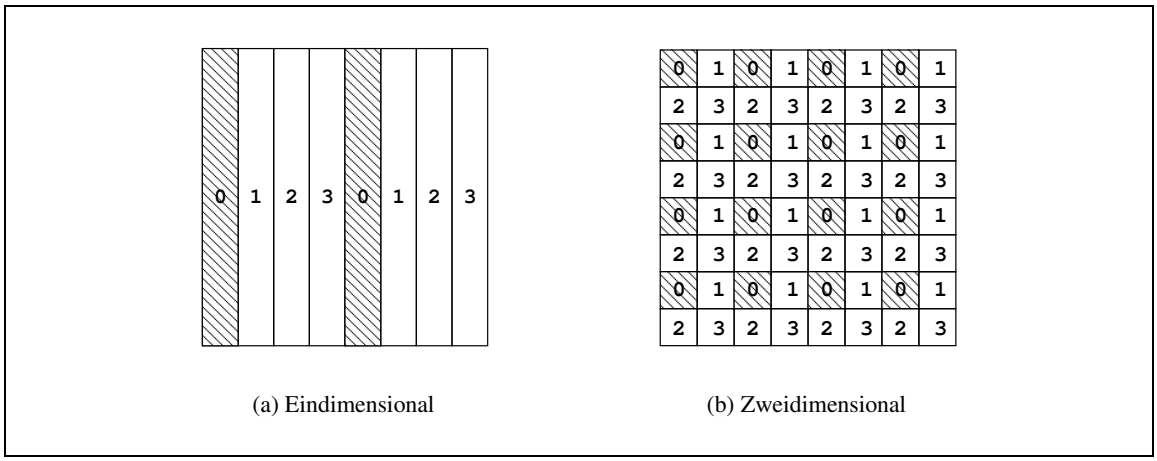


Abbildung 3.6: Blockzyklische Aufteilungen bei 4 Prozessen [23]

Abbildung 3.7 stellt dar, wie eine 5×5 Matrix mit einem 2×2 Prozess-Gitter und den Blockgrößen $nb = mb = 2$ aufgeteilt wird.

Die Datenaufteilung bei symmetrischen Matrizen ist im Fall der dichten Besetzung, genau wie bei Bandmatrizen so, dass nur die Hauptdiagonale und entweder die untere oder die obere Dreiecksmatrix von A gespeichert werden.
Bei dicht besetzten Matrizen wird mehr Speicher benötigt, als bei Bandmatrizen, da alle Matrixelemente gespeichert werden müssen.

$$\left(\begin{array}{ccc|cc} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ \hline a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{array}\right)$$

(a) 5×5 Matrix A aufgeteilt in 2×2 Blöcke

$\rightarrow$

	0			1	
0	a_{11}	a_{12}	a_{15}	a_{13}	a_{14}
	a_{21}	a_{22}	a_{25}	a_{23}	a_{24}
	a_{51}	a_{52}	a_{55}	a_{53}	a_{54}
1	a_{31}	a_{32}	a_{35}	a_{33}	a_{34}
	a_{41}	a_{42}	a_{45}	a_{43}	a_{44}

(b) Daraus entstehendes 2×2 Gitter

Abbildung 3.7: Eine 5×5 Matrix zerlegt in 2×2 Blöcke auf einem 2×2 Prozess-Gitter [23]

3.4 Prinzipielle Funktionsweise der Gleichungslöser

3.4.1 Das Cholesky Verfahren

Da A bei der FEM symmetrisch, positiv definit ist, kann zur Lösung des LGS das Verfahren von Cholesky benutzt werden [9] [11]. Es bietet einige Vorteile gegenüber anderen direkten Lösungsverfahren. Es spart Speicherplatz, da nur die Elemente der unteren Dreiecksmatrix gebraucht werden. Die mittlere Zahl an wesentlichen Rechenoperationen (Multiplikationen und Divisionen) des Verfahrens ist $1/6(n^3 + 9n^2 + 2n)$, wobei n die Problemgröße ist. Das Verfahren ist demnach asymptotisch doppelt so schnell, wie das Gauß'sche Eliminationsverfahren, dessen Rechenaufwand $1/3(n^3 + 3n^2 - n)$ beträgt [9]. Bei der Lösung ergeben sich geringere Rundungsfehler bei der „Produktsummenakkumulation".
Aus diesen Gründen basieren alle Lösungsalgorithmen von ScaLAPACK für LGS mit symmetrisch, positiv definiten Matrizen auf diesem Verfahren.

Bei dem Verfahren wird die symmetrische Matrix A zunächst zerlegt in

$$A = LL^T, \tag{3.3}$$

wobei L eine linke untere Dreiecksmatrix mit positiven Diagonalelementen l_{kk} ist. Es gilt also

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & \dots & l_{n1} \\ 0 & l_{22} & \dots & l_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & l_{nn} \end{pmatrix}$$

mit $a_{ik} = a_{ki}$.
Die Elemente von L bestimmen sich sukzessive nach der Rechenvorschrift

$$\left.\begin{array}{lll} l_{11} = & \sqrt{a_{11}} & \\ l_{ik} = & (a_{ik} - \sum_{j=1}^{k-1} l_{ij} l_{kj})/l_{kk} & k = 1, 2, ..., i-1 \\ l_{ii} = & \sqrt{(a_{ii} - \sum_{j=1}^{i-1} l_{ij}^2)} & \end{array}\right\} i = 2, 3, ..., n, \tag{3.5}$$

wobei gilt: $l_{kk} \neq 0,\ a_{11} \geq 0$.
Die Elemente werden demnach in der folgenden Reihenfolge berechnet:

$$l_{11}, l_{21}, l_{22}, l_{31}, ..., l_{33}, ..., l_{n1}, ..., l_{nn}$$

Das Verfahren von Cholesky besteht aus drei Lösungsschritten

$$A = LL^T \qquad \text{Cholesky-Zerlegung} \tag{3.6}$$
$$L\vec{c} = \vec{b} \qquad \text{Vorwärtseinsetzen} \rightarrow \vec{c} \tag{3.7}$$
$$L^T\vec{x} = \vec{c} \qquad \text{Rückwärtseinsetzen} \rightarrow \vec{x}, \tag{3.8}$$

womit das Gleichungssystem $A\vec{x} = \vec{b}$ gelöst werden kann.
Neben der angegebenen Rechenvorschrift (Gl. (3.5)) existieren auch andere Formen der Berechnung von L [12]. In ScaLAPACK wird ein paralleles Verfahren verwendet, das dem hier vorgestellten ähnelt (DPOTF2).

3.4.2 Der Löser für dicht besetzte, symmetrische, positiv definite Matrizen

In ScaLAPACK wird für diese Matrizen die Block Cholesky Zerlegung benutzt, bei der als erstes die $n \times n$ Matrizen A, L und L^T zerlegt werden und somit das Gleichungssystem

$$\begin{aligned} \begin{pmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{pmatrix} &= \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{pmatrix} \\ &= \begin{pmatrix} L_{11}L_{11}^T & L_{11}L_{21}^T \\ L_{21}L_{11}^T & L_{21}L_{21}^T + L_{22}L_{22}^T \end{pmatrix} \end{aligned} \tag{3.9}$$

aufgestellt werden kann, wobei die Block-Matrizen folgende Dimensionen besitzen:
A_{11}: $nb \times nb$, A_{21}: $(n - nb) \times nb$ und A_{22}: $(n - nb) \times (n - nb)$.
Da L_{11}, der untere Dreiecks Cholesky Faktor von A_{11} berechnet werden kann (Kap. 3.4.1), können die Block-Gleichungen umgestellt werden durch

$$\begin{aligned} L_{21} &\leftarrow A_{21}(L_{11}^T)^{-1} \\ \tilde{A}_{22} &\leftarrow A_{22} - L_{21}L_{21}^T = L_{22}L_{22}^T. \end{aligned} \tag{3.10}$$

Dieser Vorgang wird rekursiv für die Matrix $\tilde{A}_{22}$ fortgesetzt.

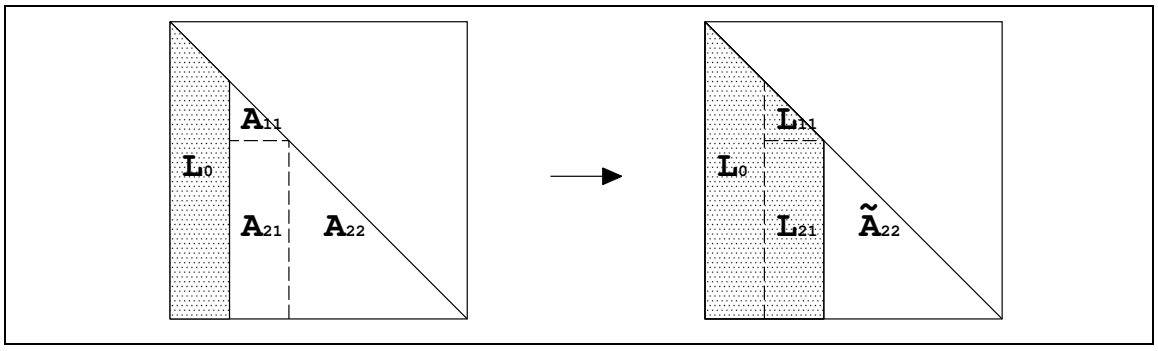


Abbildung 3.8: Die Block Cholesky Zerlegung im Verlauf einer Rechnung [28]

Abbildung 3.8 veranschaulicht, wie im Verlauf der Rechnung eine Block-Spalte $L^{(k)}$, die aus L_{11} und L_{21} besteht, berechnet wird und wie A_{22} aktualisiert wird. Die grau schraffierten Bereiche sind in dem jeweiligen Schritt bereits berechnet und müssen nicht mehr bearbeitet werden.

Die Implementierung des Algorithmus in ScaLAPACK läuft dann folgendermaßen durch Aufrufe von PBLAS Funktionen ab:

(i) Prozess P_i, der den Block A_{11} gespeichert hat, führt die Cholesky Zerlegung von A_{11} durch.

(ii) P_i sendet L_{11} an alle Prozesse der aktuellen Block-Spalte, wo anschließend jeweils die Blöcke von L_{21} berechnet werden.

(iii) Die Blöcke von L_{21} werden an alle Prozesse geschickt. Alle Prozesse haben nun ihren eigenen Teil von L_{21} und somit auch von L_{21}^T und aktualisieren ihren Teil der Matrix A_{22}.

Die Details dieser Rechenvorschrift können der folgenden Referenz entnommen werden [28].

3.4.3 Der Löser für symmetrische, positiv definite Bandmatrizen

Bei Bandmatrizen mit dünner Struktur wird ein **„Divide and Conquer“** Algorithmus angewandt. Er besteht aus drei Phasen:

(i) Das Problem wird in mehrere kleine Teilprobleme derselben Art aufgeteilt (Divide-Schritt).

(ii) Die kleineren Teilprobleme werden (größtenteils) unabhängig voneinander gelöst.

(iii) Es wird aus den Lösungen der Teilprobleme eine Lösung für das Gesamtproblem konstruiert (Conquer-Schritt).

In Phase (i) wird zunächst $A\vec{x} = \vec{b}$ von links mit einer Permutationsmatrix P multipliziert, die A so umordnet, dass die Parallelität ausgenutzt werden kann.

$$PA(P^{-1}P)\vec{x} = P\vec{b} \tag{3.11}$$

Diese Matrix wird nun durch die Cholesky Zerlegung umgeformt in

$$PAP^{-1} = LL^T. \tag{3.12}$$

Werden nun die folgenden Substitutionen durchgeführt

$$\vec{x'} = P\vec{x}, \qquad \vec{b'} = P\vec{b}, \tag{3.13}$$

ergibt sich das System

$$LL^T\vec{x'} = \vec{b'}. \tag{3.14}$$

Dies wird in Phase (ii) mit der traditionellen Methode gelöst:

$$L\vec{c} = \vec{b'}, \qquad L^T\vec{x} = \vec{c'} \tag{3.15}$$

Als letztes wird aus $\vec{x'}$ in Phase (iii) $\vec{x}$ berechnet.

$$\vec{x} = P^{-1}\vec{x'} \tag{3.16}$$

In Phase (i) wird A zerlegt in mehrere Teilmatrizen A_i, B_i, C_i und D_i (Abb. 3.9). Dies ist notwendig, da an den Überlappungsbereichen gesonderte Rechnungen stattfinden müssen, um das LGS mit der Gesamtmatrix über diese Methode parallel lösen zu können. Jeder Prozess P_i, mit Ausnahme vom letzten, hat dann die vier, ihm zugehörigen Teilmatrizen gespeichert. Anschließend werden die folgenden Berechnungen durchgeführt:
Zunächst senden alle Prozesse P_i ihre Teilmatrix D_i an Prozess P_{i+1}, wobei P_1 nur sendet und P_n nur empfängt. Dies sind Daten, die von beiden Nachbarprozessen benötigt werden. Die Teilmatrizen sind dann lokal wie in Abbildung 3.10 gespeichert. Jeder Prozess kann simultan die Cholesky Zerlegung

$$A_i = L_iL_i^T \tag{3.17}$$

durchführen, so dass A_i überschrieben wird mit L_i. Um das lokale Ergebnis $\vec{x'}$ berechnen zu können, muss die Matrix wie in Abbildung 3.11 dargestellt, aufgefüllt werden. Das ist der große Schwachpunkt des Verfahrens, da dies eine Erhöhung der Rechenzeit gegenüber dem seriellen Algorithmus zur Folge hat. Je schmaler die Bandbreite ist, umso kleiner werden die Überlappungsbereiche $B_i,\ C_i,\ D_i$ und somit auch der zusätzliche Rechenaufwand.

Auf weitere Einzelheiten der Implementierung wird hier wegen der Komplexität nicht eingegangen; sie können der folgenden Referenz entnommen werden [29].

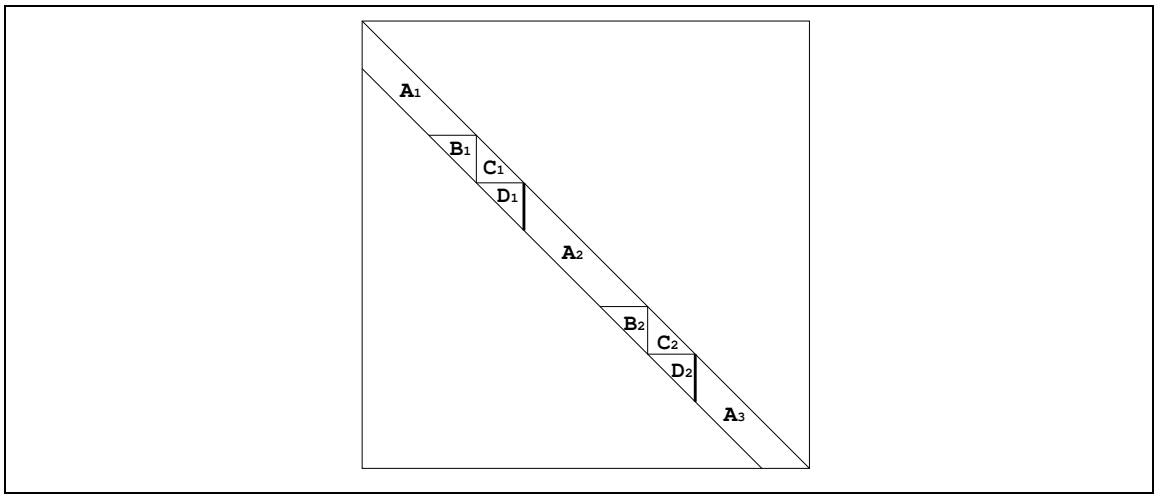


Abbildung 3.9: Divide and Conquer Aufteilung des unteren Teils einer symmetrischen Bandmatrix [29]

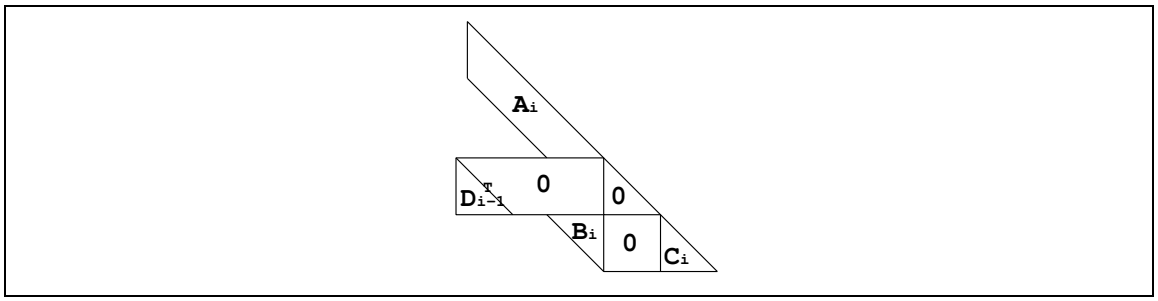


Abbildung 3.10: Lokale Matrix auf Prozess P_i nach der Initialkommunikation von D_i [29]

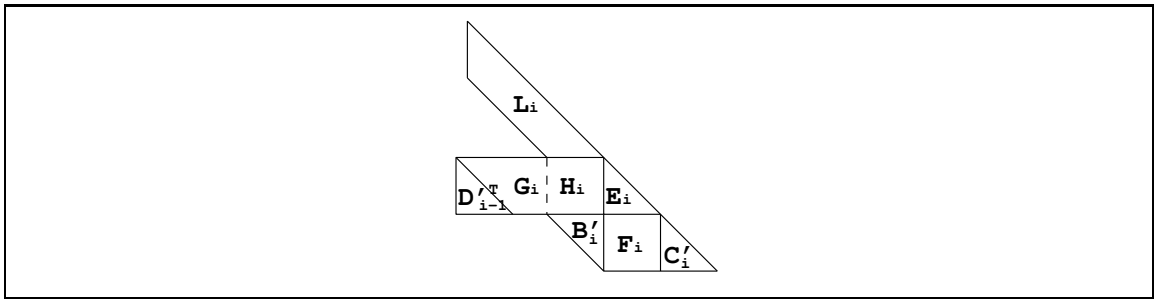


Abbildung 3.11: Lokale Matrix auf Prozess P_i nach Beendigung von Phase (i) [29]

3.5 Messungen zur Leistungsfähigkeit der ScaLAPACK Routinen

Im folgenden wird untersucht, wieviel Rechenzeit und Speicherplatz von den ScaLAPACK Routinen zur Lösung von LGS benötigt wird. Durch unterschiedliche Wahl der Einträge der Matrix oder der RHS ändert sich der Rechenaufwand bei den hier benutzten direkten Lösungsverfahren nicht. Daher werden beliebige Einträge für die RHS gewählt. Als Matrizen werden Beispiele vom Matrix Market [30] benutzt, die bei Finite-Elemente-Berechnungen entstanden sind. Die einzigen Matrixparameter, die die Messergebnisse beeinflussen sind Dimension und Bandbreite, die bei den für die Rechnungen verwendeten Matrizen variieren. Der Löser für dicht besetzte Matrizen wird in diesem Kapitel mit Löser A und der für Bandmatrizen mit Löser B bezeichnet.
Wie in Kapitel 3.3 beschrieben, werden die Blockgrößen für Löser A vorgegeben. Diese werden für Spaltenaufteilung (nb) und für Zeilenaufteilung (mb) gleich gewählt, so dass quadratische Blöcke entstehen. Da die Matrizen ebenfalls quadratisch sind, führt dies zu einer effizienten Datenkommunikation.

3.5.1 Einfluss von physikalischer Prozessor- und virtueller Prozesstopologie

Generell ist bei parallelen Rechnungen zu erwarten, dass die Rechenzeit mit steigender Prozesszahl sinkt, da sich dann die notwendige Rechenarbeit für jeden einzelnen Prozess verringert. Das muss jedoch nicht immer gelten.
Dies kann zum einen an der physikalischen Prozessortopologie liegen. Bei SMP-Rechnern nutzen die Prozessoren eines SMP-Knotens einen gemeinsamen Speicher, so dass hier beim Datenaustausch untereinander die Kommunikation über das Verbindungsnetzwerk nicht notwendig ist. Dies würde nur unnötig die Kommunikationswege blockieren, insbesondere da über eine Knotenverbindung Daten gleichzeitig gesendet und empfangen würden. Genau dieser Effekt tritt jedoch beim ZAMpano auf, da die zurzeit installierte MPI-Version ausschließlich Systeme mit verteiltem Speicher unterstützt.
Beim Programmaufruf auf dem ZAMpano kann gezielt gesteuert werden, ob z.B. vier Prozessoren eines SMP-Knotens oder vier Knoten mit je einem Prozessor genutzt werden sollen. Im ersten Fall befinden sich alle Prozessoren auf einem SMP-Knoten, so dass jede Datenkommunikation über eine Leitung zum Verbindungsnetzwerk und zurück stattfindet. Im zweiten Fall sind alle Prozessoren auf verschiedenen SMP-Knoten, wodurch diese Art der Kommunikationsblockierung nicht gegeben ist. Die letztere Alternative bringt demnach geringere Rechenzeiten (Tab. 3.1). Um die Programmaufrufe einheitlich zu halten, werden diese immer so getätigt, dass möglichst wenige Ressourcen, also SMP-Knoten belegt werden, da die Zunahme der Rechenzeit verhältnismäßig gering ist. Das bedeutet, dass möglichst viele Prozessoren eines SMP-Knotens benutzt werden, da diese ansonsten auch nicht für andere Zwecke verwendet werden können. Falls beispielsweise vier SMP-Knoten mit je einem Prozessor für eine Rechnung benutzt werden, sind die restlichen neun Prozessoren dieser Knoten ebenfalls blockiert. Die Belegung der Prozessoren kann auf dem ZAMpano nur so gemacht werden, dass auf jedem SMP-Knoten die gleiche Anzahl an Prozessoren verwendet wird. Dies bedeutet beispielweise bei sieben Prozessen, dass auch sieben SMP-Knoten verwendet werden müssen, da diese Zahl eine Primzahl ist. Bei elf Prozessen hieße dies, dass theoretisch auch elf SMP-Knoten genutzt werden müssen. Diese Menge ist auf dem ZAMpano jedoch nicht vorhanden. Je nachdem, mit wievielen Prozessen das Programm gestartet werden soll, ist somit eine effiziente Ausnutzung der Ressourcen nicht möglich oder das Programm kann mit dieser Prozesszahl gar nicht durchlaufen werden (Abb. 3.12). Bei den Auswertungen wird dann eventuell eine Zunahme der Rechenzeit beim Übergang von einem Ressourcen verschwenderischen Aufruf zu einem sparenden sichtbar und umgekehrt. Ein Beispiel für eine geringere, zu erwartende Verbesserung oder sogar Verschlechterung der Rechenzeit wäre der Übergang von sieben nach acht Prozessen. Bei sieben Prozessen wird nur ein Prozessor je SMP-Knoten benutzt, bei acht Prozessen alle vier.
Auf der CRAY T3E ist dieses Problem nicht gegeben, da bei diesem System die Geschwindigkeiten der Speicherzugriffe unabhängig von den verwendeten Prozessoren sind.

SMP-Knoten	Prozessoren je SMP-Knoten	Rechenzeit [s] einer Matrix mit	
		$n = 4884$, $\beta = 141$ (Abb. 3.14)	$n = 11948$, $\beta = 1244$ (Abb. 3.16)
1	4	2,06	313
2	2	1,93	291
4	1	1,84	271

Tabelle 3.1: Einfluss des physikalischen Prozessorgitters des ZAMpano am Beispiel des Lösers für Bandmatrizen auf die Rechenzeit

Ein weiterer Effekt ist die virtuelle Prozesstopologie bei dicht besetzten Matrizen. Auch hier ist je nach Prozesszahl eine mehr oder wenige effiziente Gittereinteilung und somit Datenaufteilung möglich (Kap. 3.2). Ein optimales Gitter für Löser A liegt vor, wenn gilt: $P_r = P_c$. Je größer der verhältnismäßige Unterschied zwischen den beiden Größen ist, umso schlechter ist das Gitter (Abb.

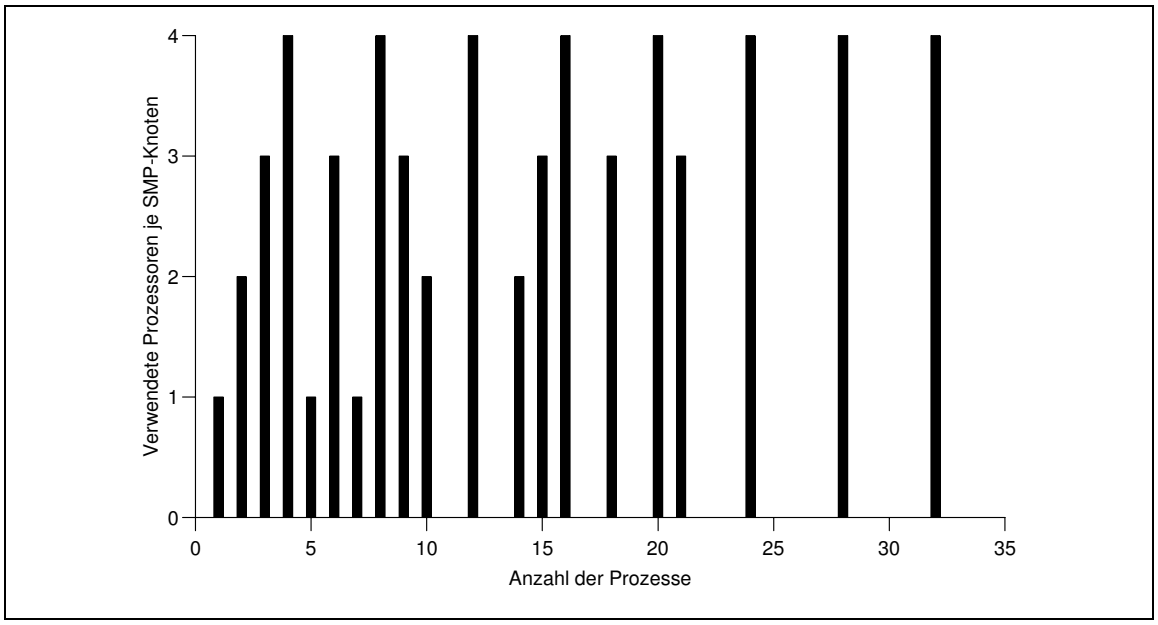


Abbildung 3.12: Anzahl verwendeter Prozessoren je SMP-Knoten auf dem ZAMpano in Abhängigkeit von der Prozesszahl

3.13). Da dieses Gitter bei Löser B eindimensional ist, ist das Problem für diesen Löser nicht gegeben.

Tabelle 3.2 belegt, dass ein quadratisches virtuelles Prozessgitter die Rechenzeit für eine konstante Prozesszahl minimiert. Es werden verschieden große Matrizen mit Löser A und 16 Prozessen benutzt. Dies ist die maximale Anzahl an Prozessen, bei der die Erzeugung eines quadratischen Gitters auf dem ZAMpano möglich ist. So kann die Rechnung mit möglichst vielen Kombinationen der Dimensionierung des Gitters ausgeführt werden. Je größer die Abweichung zwischen Prozessspalten und Prozessreihen ist, umso länger dauert es, bis ein Ergebnis vorliegt. Ist das Prozessgitter nicht quadratisch, unterscheiden sich die Rechenzeiten geringfügig in Abhängigkeit davon, ob ein eher „flaches“ oder „hohes“ Gitter gewählt wird, d.h. ob das Gitter aus mehr Prozessspalten als Prozessreihen besteht oder umgekehrt. Es lässt sich aber keine allgemeine Aussage darüber machen, welches besser ist. Für die Messungen in Kapitel 3.5.2 wird das Gitter so gewählt, dass $P_c \geq P_r$ gilt.

P_r	P_c	Rechenzeit [s] einer Matrix mit	
		$n = 4884$, $\beta = 141$ (Abb. 3.14)	$n = 11948$, $\beta = 1244$ (Abb. 3.16)
1	16	25,5	220
2	8	17,9	171
4	4	15,2	156
8	2	17,8	187
16	1	25,5	220

Tabelle 3.2: Einfluss der Dimensionierung des virtuellen Prozessgitters beim Löser für dicht besetzte Matrizen auf die Rechenzeit

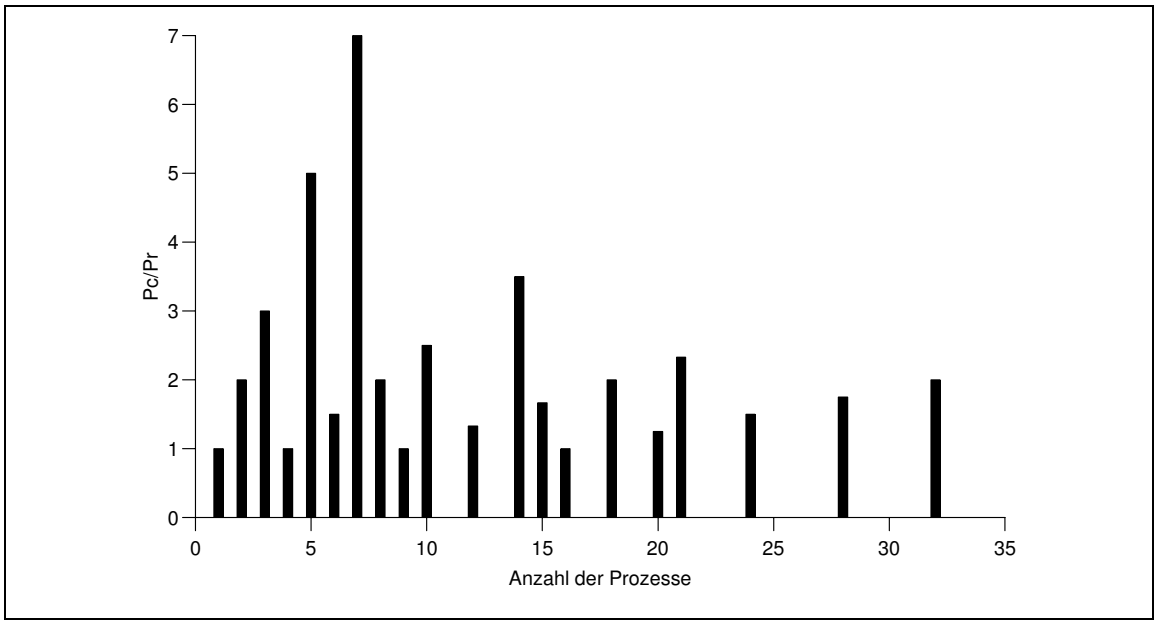


Abbildung 3.13: Verhältnis zwischen Prozessreihen und Prozessspalten des verwendeten Prozessgitters in Abhängigkeit von der Prozesszahl

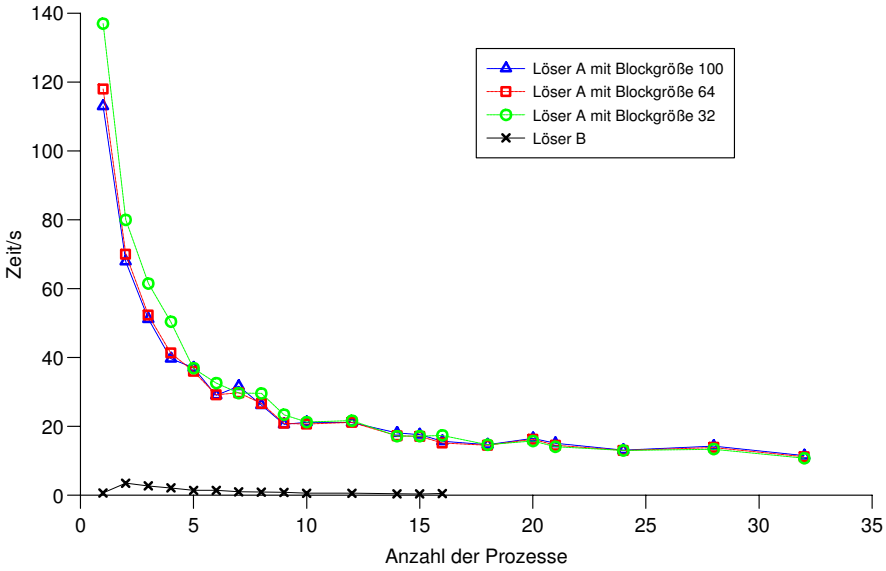
3.5.2 Messergebnisse

In den folgenden Diagrammen werden die Rechenzeiten als Funktion der Prozesszahl, ohne Bearbeitungszeit für Ein-/Ausgabe und Ergebnisüberprüfung, dargestellt. Da die Untersuchungen primär auf dem ZAMpano stattfinden, wird bei der Ergebnisdiskussion zu diesem System kein expliziter Hinweis darauf gegeben, wenn diese sich auf das ZAMpano bezieht. Handelt es sich hingegen um Erläuterungen zu Messungen auf der CRAY T3E, wird dies entsprechend angegeben. Bei Löser A werden Programmdurchläufe mit unterschiedlichen Blockgrößen durchgeführt, da diese die Leistungsfähigkeit beeinflusst (Kap. 3.3.2).

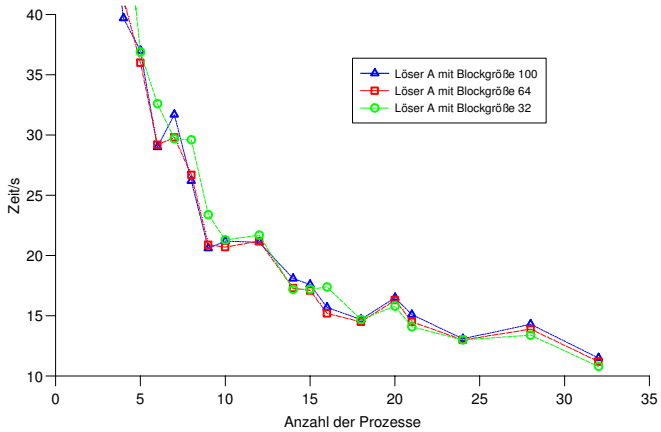
Bei einer Matrix mit den Parametern $n = 4884$ und $\beta = 141$ benötigt Löser B wesentlich weniger Rechenzeit, als Löser A, da die Bandbreite β hier relativ klein ist (Abb. 3.14 (a)).
Die Matrix selbst hat aber eine so geringe Dimension, dass Löser B nur mit maximal 16 Prozessen ausgeführt werden kann (Abb. 3.14 (c)). Ansonsten ist durch die untere Schranke für die Blockgröße eine konstante Rechenzeit zu erwarten, da dann die zusätzlichen Prozesse ungenutzt bleiben (Kap. 3.3.1). Die mit 16 Prozessen benötigte Rechenzeit bringt keine Verbesserung gegenüber dem seriellen Algorithmus (parallelen Algorithmus mit einem Prozess). Beim Übergang von einem nach zwei Prozessen steigt die Rechenzeit um das 5-fache an. Dies liegt an der notwendigen Auffüllung der Matrix des parallelen Algorithmus, was eine Erhöhung des Rechenaufwandes gegenüber dem vergleichbaren seriellen Algorithmus bringt (Kap. 3.4.3). Die virtuelle Prozesstopologie hat hier keinen Einfluss auf die Messergebnisse, da sie eindimensional ist. Die Auswirkungen der physikalischen Prozessortopologie macht sich besonders bei 6, 8, 12 und 16 Prozessen bemerkbar. Hier kommt es nur zu einer sehr geringen Abnahme der Rechenzeit bzw. bei 16 Prozessen sogar zu einer Zunahme. An diesen Stellen werden im Vergleich zu den vorigen Messwerten mehr Prozessoren je SMP-Knoten verwendet, wodurch mehr Kollisionen bei der Datenkommunikation entstehen.
Bei Löser A können unabhängig von den Matrixparametern immer alle 32 Prozesse ausgenutzt werden (Abb. 3.14 (b)). Wie bei Löser B bewirkt die physikalische Prozessortopologie eine Verschlechterung der Leistungsfähigkeit bei 8, 12, 16 und zusätzlich 20 Prozessen. Bei 6 Prozessen ist dieser Effekt bei Löser A nicht zu erkennen, da hier eine bessere virtuelle Prozesstopologie im Vergleich zur vorigen Messstelle entgegen wirkt. Aus demselben Grund ist die Verschlechterung der Leistungsfähigkeit bei 16 Prozessen hier nicht so extrem wie bei Löser B. Obwohl bei 8, 12 und 20 Prozessen dasselbe zu erwarten wäre, ist hier keine Verbesserung feststellbar. Da die

exakten Kommunikationsvorgänge bei der Rechnung und die jeweilige Beeinflussung von physikalischer Prozessor- und virtueller Prozesstopologie nicht ohne enorm großen Aufwand nachzuvollziehen sind, ist eine quantitative Analyse der Zu- bzw. Abnahme der Rechenzeit nicht ohne weiteres durchführbar.
Je mehr Prozesse genutzt werden, umso kleiner sollte die Blockgröße gewählt werden, um eine gute Verteilung der Datenlast zu gewährleisten. Entgegengesetzt sollte bei kleiner Prozesszahl eine hohe Blockgröße genutzt werden, um die notwendige Datenkommunikation minimal zu halten. Aus diesem Grund bringt eine Blockgröße von 100 die geringsten Rechenzeiten von Löser A mit bis zu 4 Prozessen. Bei einer Prozesszahl von 5 bis 18 ist eine Blockgröße von 64 am besten geeignet. Ab 20 Prozessen sind die Rechenzeiten mit einer Blockgröße von 32 am niedrigsten. Dies bestätigt die allgemeine Empfehlung eine Blockgröße von 64 zu benutzen, die dann je nach Parameter angepasst werden kann [23].

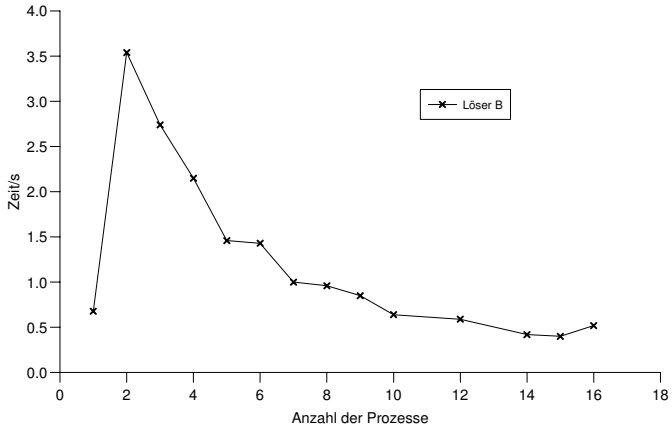
Auf der CRAY T3E liegen die Rechenzeiten der seriellen Löser wegen der höheren Prozessorleistung unter denen des ZAMpano. Bei Löser A beträgt der Unterschied etwa 40 %, bei Löser B nur 20 %. Die physikalische Prozessortopologie hat keinen Einfluss auf die Messergebnisse der parallelen Rechnungen auf der CRAY T3E, da bei diesem System die Geschwindigkeiten der Speicherzugriffe unabhängig von den verwendeten Prozessoren sind. Betrachtet man die Messungen der selben Matrix auf dieser Maschine, verlaufen die Kurven wesentlich glatter (Abb. 3.15).
Nur bei Löser A wird der Verlauf durch die virtuelle Prozesstopologie beeinflusst (Abb. 3.15 (b)). Eine schlechtere Dimensionierung des Prozessgitters im Vergleich zum vorigen Messwert wird bei 14 und 28 Prozessen sichtbar, eine entsprechende Verbesserung bei 12 und 16 Prozessen. Der Verlauf der einzelnen Kurven von Löser A mit unterschiedlichen Blockgrößen unterscheidet sich von denen auf dem ZAMpano. Um auf der CRAY T3E diesbezüglich möglichst geringe Rechenzeiten zu erhalten, muss mehr Wert auf eine gute Datenverteilung, statt auf eine geringe Kommunikation gelegt werden, da das Verbindungsnetzwerk wesentlich leistungsfähiger ist. Das bedeutet es sollte eine kleinere Blockgröße gewählt werden. So liegen die Rechenzeiten bei einer Blockgröße von 100 immer über den Rechenzeiten mit einer Blockgröße von 64 oder 32. Bis zu einer Prozesszahl von 12 ist eine Blockgröße von 64 am besten geeignet, ansonsten werden mit einer Blockgröße von 32 bessere Rechenzeiten erzielt.
Die Zunahme der Rechenzeit von Löser B mit 2 Prozessen ist mit Faktor 2,6 auf der CRAY nur etwa halb so groß, wie auf dem ZAMpano (Abb. 3.15 (c)). Dies liegt ebenfalls an dem besseren Verbindungsnetzwerk. Sowohl Latenzzeit, als auch Bandbreite besitzen etwa 10 mal so gute Werte, wie das ZAMpano (Tab 2.1). Aus diesem Grund kommt dieser Unterschied zustande, obwohl die Kommunikation in diesem Beispiel je nach Prozesszahl nur etwa 15 % der Rechenzeit ausmacht. So kann schon ab 5 Prozessen eine geringere Rechenzeit erzielt werden, als mit einem Prozess.



(a) Gesamtes Diagramm

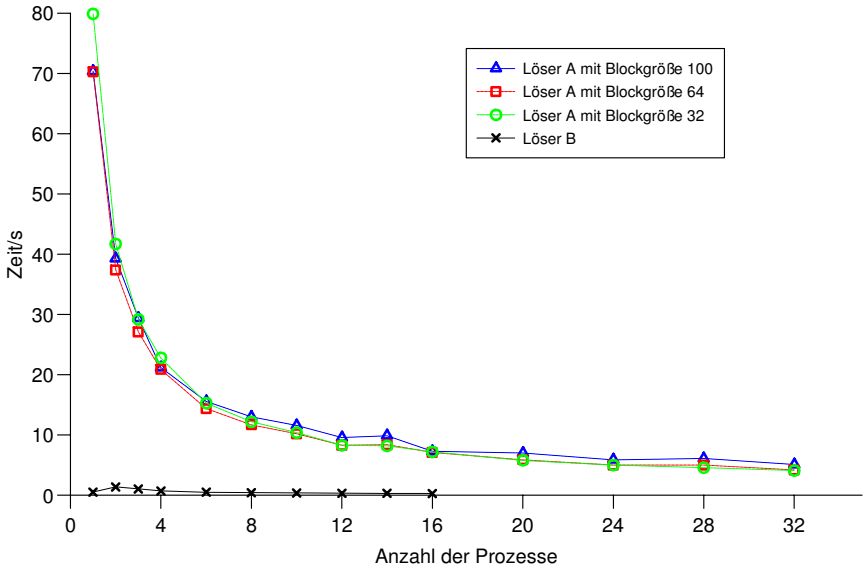


(b) Diagrammausschnitt, nur Löser A

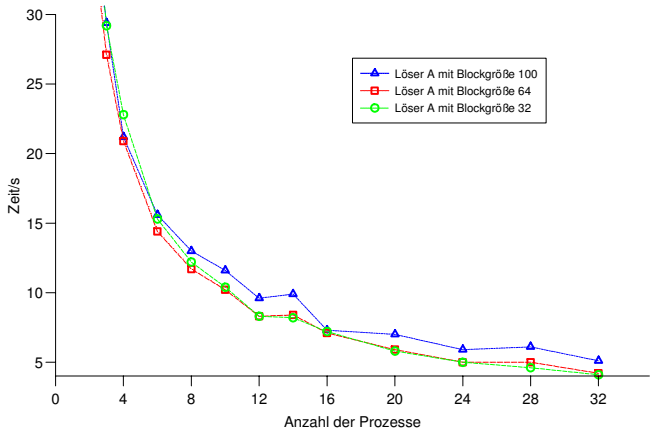


(c) Diagrammausschnitt, nur Löser B

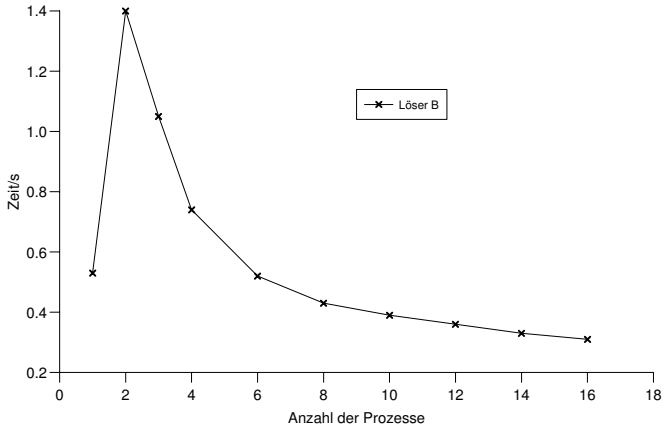
Abbildung 3.14: Rechenzeiten als Funktion der Prozesszahl für eine Matrix der Dimension $n = 4884$ mit Bandbreite $\beta = 141$ auf dem ZAMpano



(a) Gesamtes Diagramm



(b) Diagrammausschnitt, nur Löser A



(c) Diagrammausschnitt, nur Löser B

Abbildung 3.15: wie Abb. 3.14, auf der CRAY T3E

In Abbildung 3.16 wird ein LGS mit einer Matrix mit den Parametern $n = 11948$ und $\beta = 1244$ untersucht. Die Matrix hat sowohl eine höhere Dimension als auch eine größere Bandbreite, wobei die Vergrößerung der Bandbreite noch stärker ist. Aus diesem Grund ist die Zunahme der Rechenzeit bei 2 Prozessen mit Faktor 9 hier noch größer, als in dem vorigen Beispiel. Durch die große Bandbreite kann Löser B nur mit bis zu 5 Prozessen genutzt werden. Ab 12 Prozessen wird diese Rechenzeit von Löser A unterboten. Sie ist aber bei Nutzung der maximalen Anzahl an Prozessen des ZAMpano immer noch höher als die von Löser B bei einem Prozess. Da bei dieser Matrix wesentlich mehr Daten verarbeitet werden müssen, als bei der vorigen, sollte die Wahl der Blockgröße bei Löser A entsprechend erhöht werden. Von den drei getesteten Blockgrößen eignet sich 100 für alle möglichen Prozesszahlen am besten.

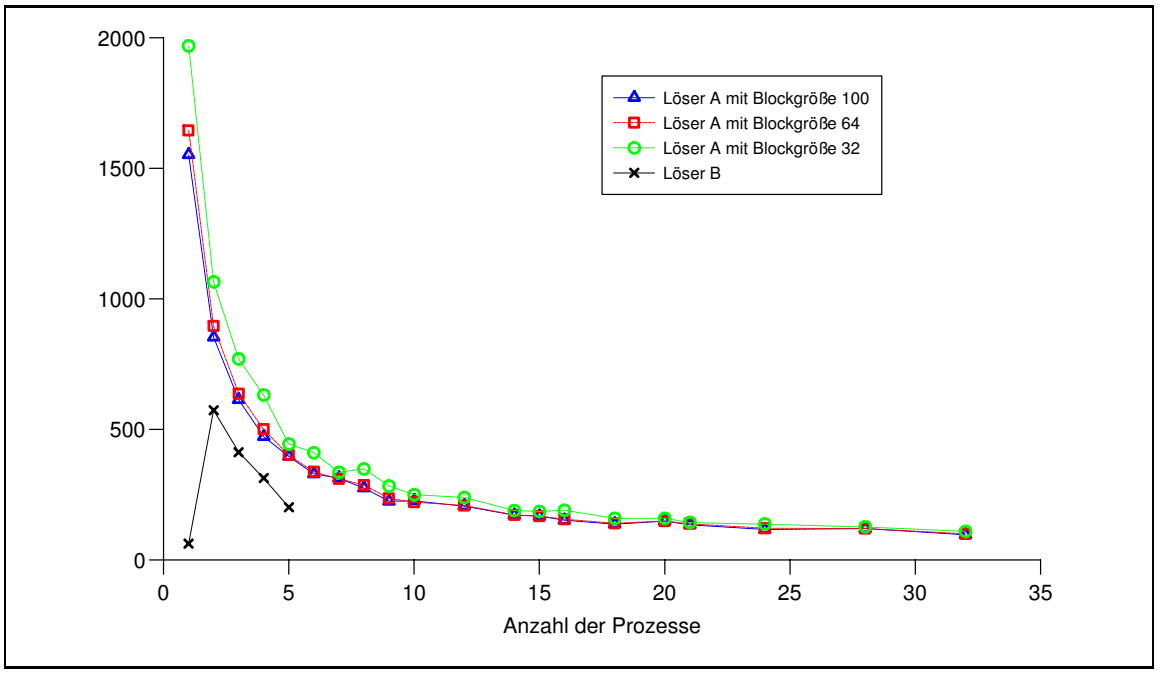


Abbildung 3.16: wie Abb. 3.14, mit $n = 11948$ und $\beta = 1244$

Für eine Matrix mit einer sehr hohen Dimension $n = 90449$ und relativ geringen Bandbreite $\beta = 614$ sollte Löser B sehr gut geeignet sein (Abb. 3.17). Die Rechenzeit bei der maximalen Anzahl an Prozessen entspricht aber wieder etwa der mit einem Prozess. Bei 2 Prozessen ist die Zunahme der Rechenzeit mit Faktor 8 nicht ganz so hoch wie im vorigen Beispiel, da trotz gestiegener Matrixdimension die Bandbreite wesentlich kleiner ist. Löser A kann zur Berechnung des LGS nicht verwendet werden, da die Matrix zu groß ist. Löser A benötigt ausreichend Speicherplatz für die Matrix, um alle $n \times n$ Elemente speichern zu können. Löser B genügt eine Kapazität für $n \times (\beta + 1)$ Elemente. Auffällig ist die starke Zunahme der Rechenzeit bei 4 und 6 Prozessen. Weniger hohe Zunahmen konnten mit 6 Prozessen ebenso bei der Matrix aus Abbildung 3.14 festgestellt werden. Mit 4 Prozessen ist dies nicht so. Vermutlich finden gerade bei großen Matrizen häufig Kollisionen bei der Datenkommunikation auf einem SMP-Knoten statt, so dass hier gerade bei der Nutzung nur eines SMP-Knotens mit allen vier Prozessoren die Rechnung mehr Zeit benötigt. Genauere Analysen dazu werden in Kapitel 4.2 vorgenommen.

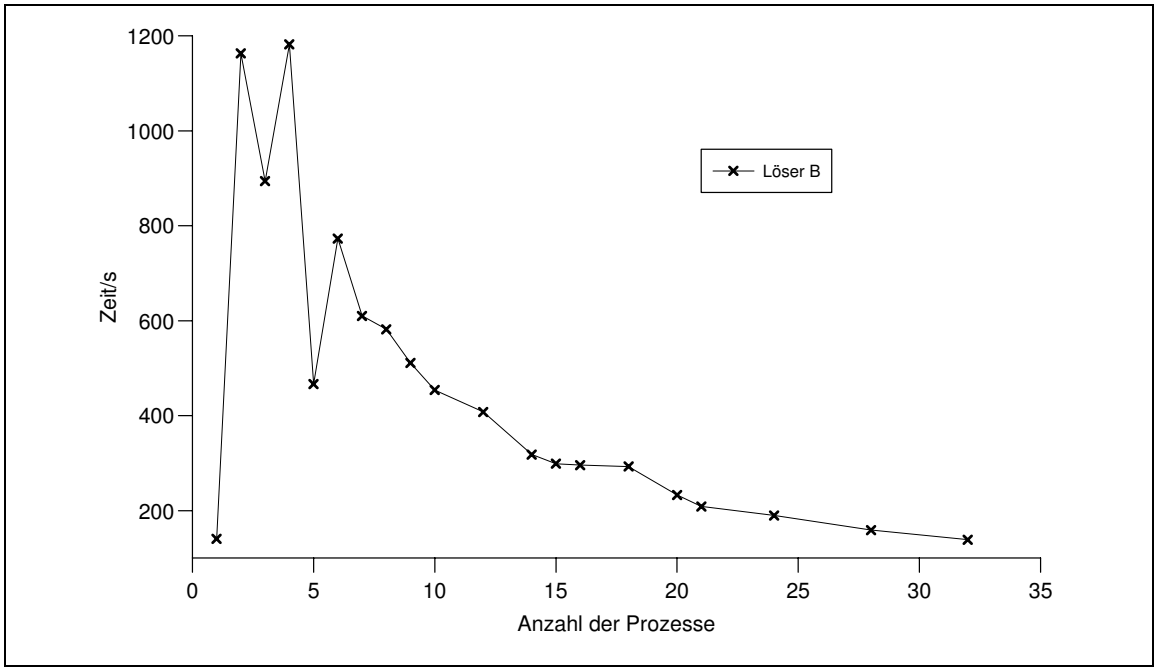


Abbildung 3.17: wie Abb. 3.14, mit $n = 90449$ und $\beta = 614$

Abbildung 3.18 und 3.19 belegen, dass der Algorithmus des Bandlösers von einer sehr schmalen Bandbreite profitiert. Es müssen dann wesentlich weniger Daten zwischen den einzelnen Prozessen ausgetauscht werden, da sich so die Datenmenge, die von jeweils zwei Nachbarprozessen zur Lösung des LGS benötigt wird, verringert (Kap. 3.4.3). Auf dem ZAMpano kann die Rechnezeit des seriellen Lösers schon mit 5 Prozessen unterboten werden. In den vorigen Beispielen gelang selbst mit 32 Prozessen nur eine Übereinstimmung der Rechenzeiten. Auf der CRAY T3E ist von einer Steigerung der Rechenzeit bei 2 Prozessen gar nichts mehr zu erkennen.

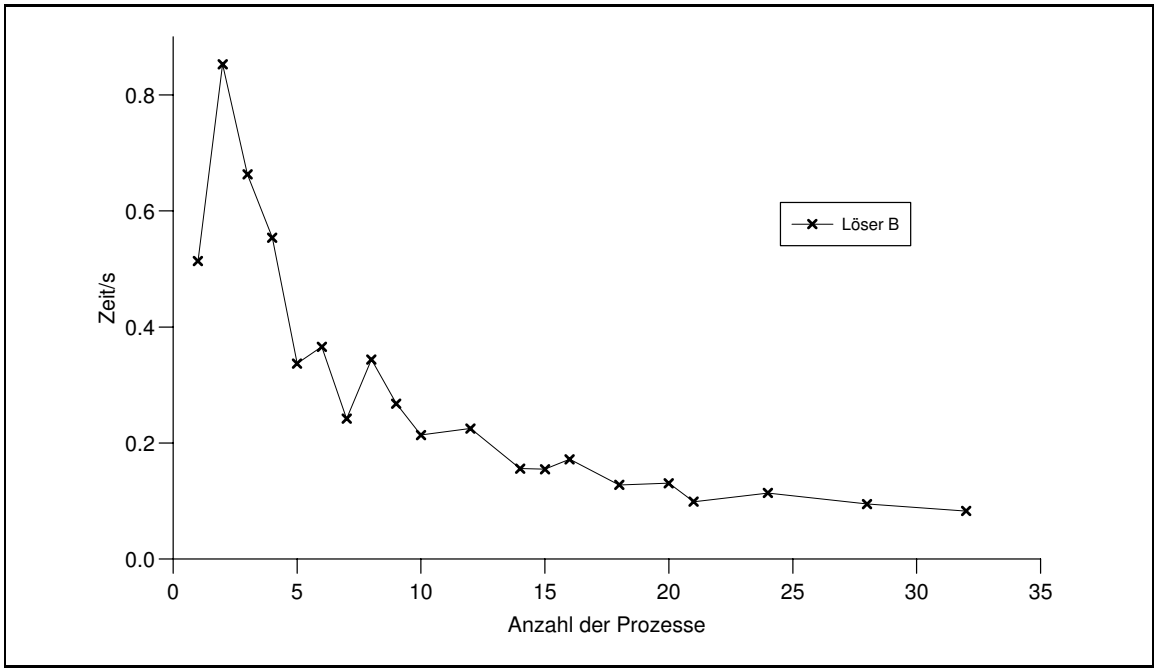


Abbildung 3.18: wie Abb. 3.14, mit $n = 100000$ und $\beta = 10$

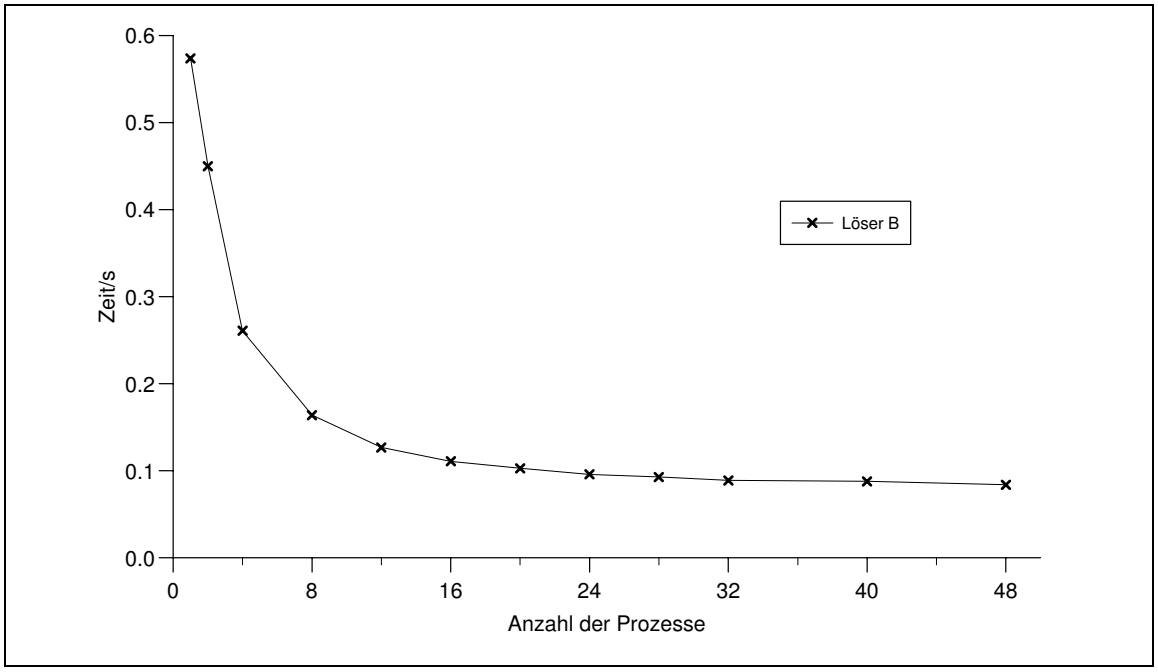


Abbildung 3.19: wie Abb. 3.14, mit $n = 100000$ und $\beta = 10$, auf der CRAY T3E

3.5.3 Speedup Vergleich: ZAMpano - CRAY T3E

Der Speedup ist ein Maß für einen parallelen Algorithmus. Er gibt die Beschleunigung des parallelen Programms gegenüber dem seriellen an.

$$S(p) = \frac{T(1)}{T(p)} \tag{3.18}$$

ist der Speedup eines parallelen Programms auf p Prozessen, wobei $T(1)$ die serielle Ausführungszeit und $T(p)$ die gemessene Ausführungszeit eines Programms bei Verwendung von p Prozessen ist.
Um die Messungen mit einer hohen Anzahl an Prozessen durchführen zu können, wird eine Matrix mit großer Dimension und sehr kleiner Bandbreite genutzt. Die Bandbreite ist so klein, dass das LGS auf der T3E auch mit einem Prozess berechnet werden kann, da dann dort weniger Speicherplatz zur Verfügung steht, als auf dem ZAMpano (Kap. 2.2.3).
Abbildung 3.20 und 3.21 zeigen den Speedup von Löser B auf dem ZAMpano bzw. auf der CRAY T3E. Beim Übergang von einem nach zwei Prozessen kommt es zunächst zu einer negativen, danach zu einer positiven, linearen Beschleunigung. Der Verlauf der Kurve fluktuiert wegen des Einflusses der physikalischen Prozessortopologie (Kapitel 3.5.1). Auf der CRAY hingegen ist er sehr gleichmäßig, da jeder Prozessor einen eigenen Speicherbereich hat. Werden mehr als 32 Prozesse genutzt, nimmt die Beschleunigung ab, da mit zunehmender Prozesszahl der Aufwand für die Datenkommunikation steigt. Irgendwann nimmt der Nachrichtenaustausch mehr Zeit in Anspruch, als die Rechnung selbst.

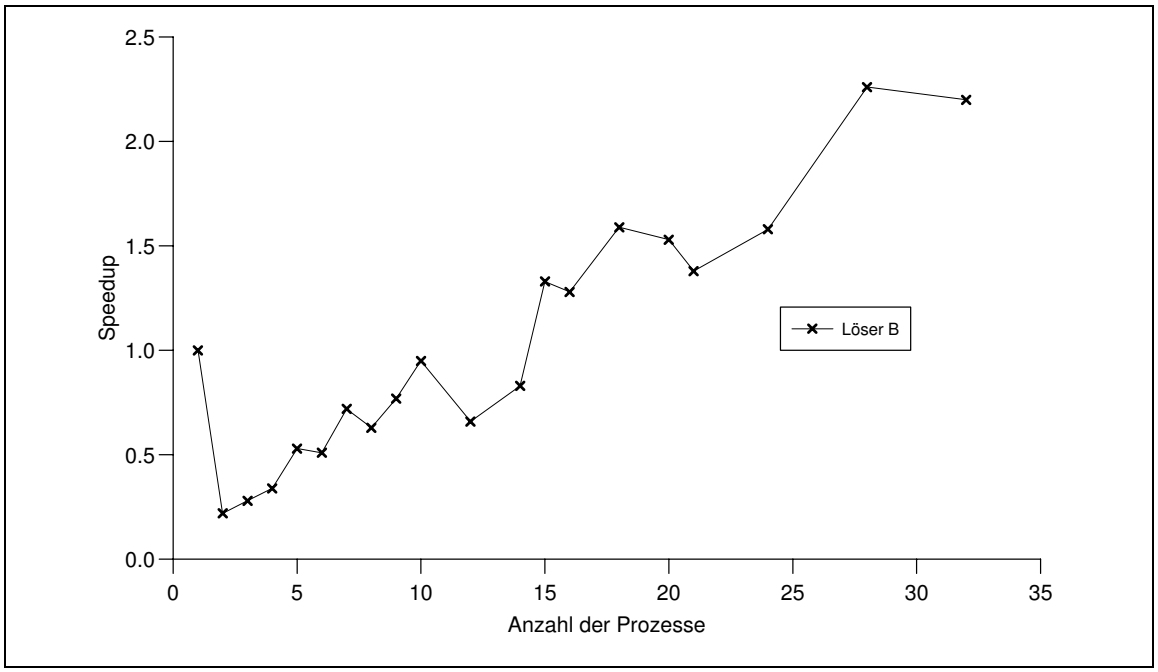


Abbildung 3.20: Speedup als Funktion der Prozesszahl für eine Matrix der Dimension $n = 20000$ mit Bandbreite $\beta = 100$ auf dem ZAMpano

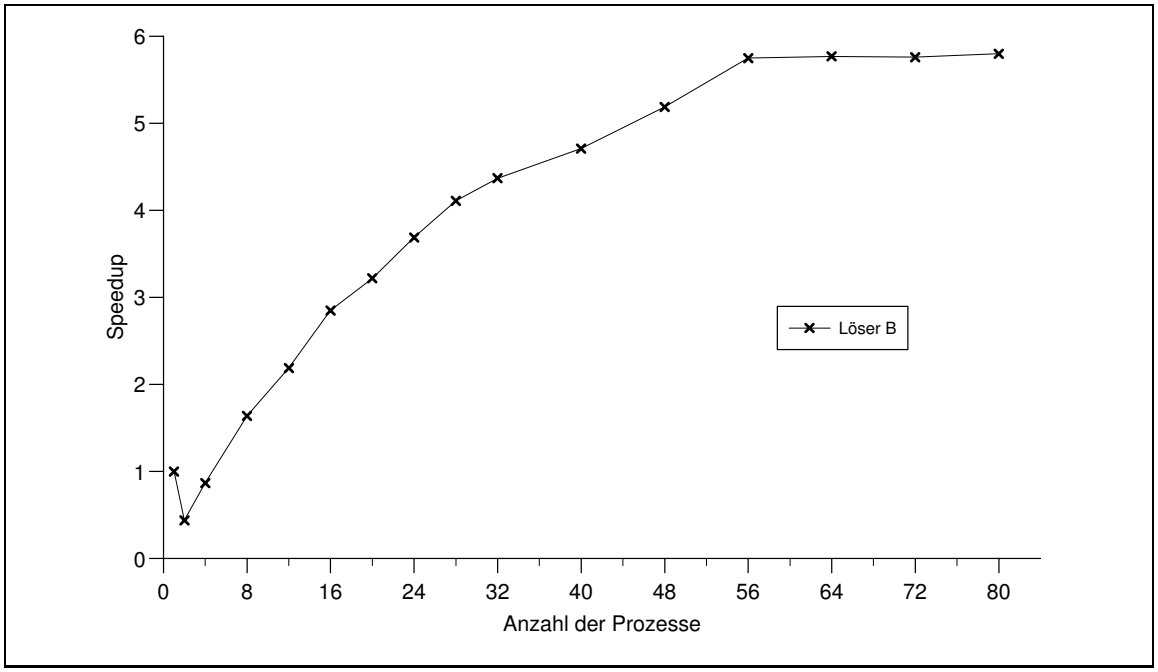


Abbildung 3.21: wie Abb. 3.20, auf der CRAY T3E

Kapitel 4

Das FEM Programm FINEART

4.1 Funktion und Aufbau von FINEART

FINEART steht für „FINite Element analysis using Adaptive Refinement Techniques“ [31] [32] [33]. Es ist ein FEM-Software-Paket, das vom „Structural Engineering Research Centre“[3] (SERC), Chennai (Madras), Indien entwickelt wurde. Es dient zur iterativen Berechnung von FEM-Problemen bis zu einer vorgegebenen Fehlertoleranz mittels h-adaptiver Netzverfeinerung. Dabei werden die Knoten nach jedem Iterationsschritt teilweise neu nummeriert.
Jeder Iterationsschritt besteht aus fünf Teilen, die in FINEART durch einzelne Module implementiert sind:

(i) Berechnung der Element-Steifigkeitsmatrizen.

(ii) Assemblierung der globalen Steifigkeitsmatrix.

(iii) Lösen des LGS.

(iv) Fehlerabschätzung.

(v) Falls der Fehler η (Gl. 2.36)) größer als die vorgegebene Toleranz η_{Ziel} ist:
Adaptive Netzverfeinerung und Start des nächsten Iterationsschrittes bei (i)

Die aktuelle Version 1.2 benutzt einen public domain Gleichungslöser von LASpack [34], der hier als Black-Box behandelt wird. Es ist ein serieller, iterativer Löser für dünn besetzte Matrizen.

4.1.1 Ein Beispiel: Die L-Domäne

Ein Beispiel, mit dem die Funktionsweise von FINEART dargestellt werden kann, ist die L-Domäne. Es ist ein zweidimensionales Gebilde in Form eines „L“, das sich aus drei Quadraten zusammensetzt. Jedes Quadrat besteht hier vor der Netzverfeinerung aus vier finiten Vierecks-Elementen mit jeweils vier Knoten (bi-linearer Verschiebungsansatz) (Abb. 4.1).
Um kinematische Bestimmtheit zu erreichen, wird die rechte Seite des „L“ in x-Richtung und die obere Seite in y-Richtung mit Gleitlagern gefesselt. Die Elementknoten, die sich an diesen Seiten befinden, haben durch die eingeschränkte Bewegungsmöglichkeit einen Freiheitsgrad weniger. Greifen an der linken Seite der Struktur Zugkräfte F an, treten Verschiebungen auf. Die L-Domäne besteht aus einem fiktiven Material, das durch die Größen Elastizitätsmodul E, Poisson-Zahl bzw. Querkontraktionszahl ν und Dichte ρ beschrieben wird. Die Werte sind in diesem Beispiel gegeben durch

$$F = 1\ N \text{ (je Elementseite)} \qquad E = 10^5\ \frac{N}{mm^2} \qquad \nu = 0,3 \qquad \rho = 1\ \frac{kg}{dm^3}. \tag{4.1}$$

[3] URL: `http://www.sercm.org/`

Es handelt sich um Materialwerte, die in der Realität häufig wiederzufinden sind. Beispielsweise Kupfer ist gegeben durch $E = 10^5$ bis $1,3 \cdot 10^5 \ \frac{N}{mm^2}$, $\nu = 0,34$, $\rho = 8,96 \ \frac{kg}{dm^3}$.
Die Seitenlänge eines finiten Elementes beträgt $25 \ mm$ und die Scheibendicke $1 \ mm$. Daraus folgt, dass in diesem Beispiel eine Flächenlast von $F = 0,04 \ \frac{N}{mm^2}$ wirkt.

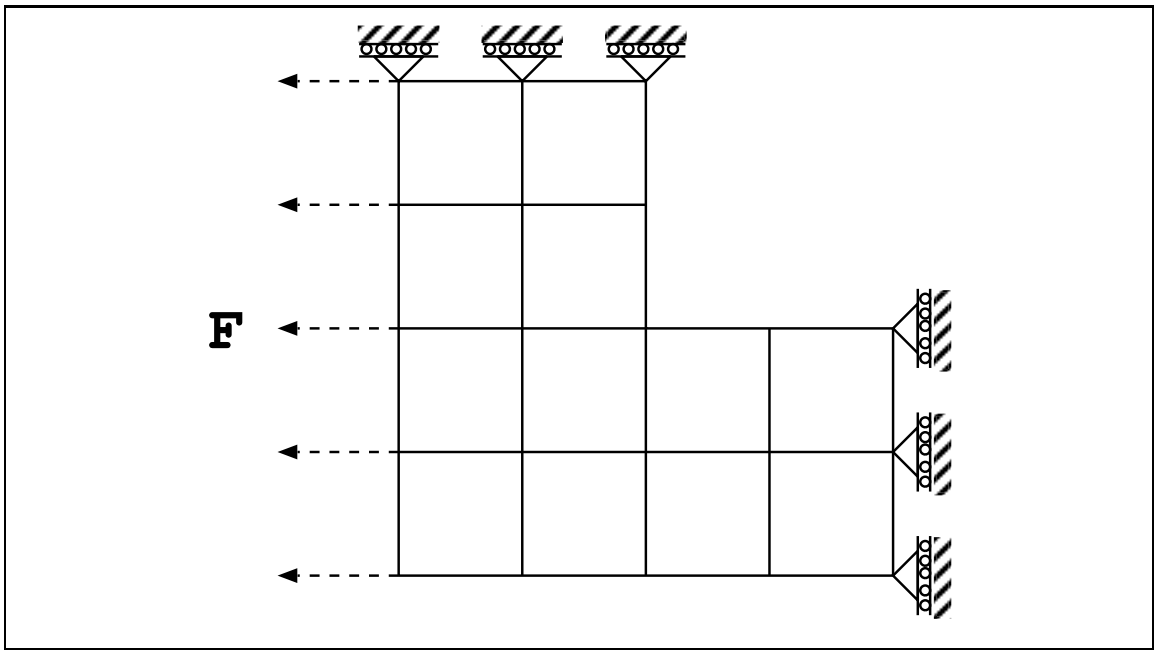


Abbildung 4.1: L-Domäne, Iterationsschritt 0 mit Lagerbeschreibung

Die Abbildungen 4.2 bis 4.5 illustrieren die Netzverfeinerung, die mit jeder Iteration besonders im Bereich der Kerbe zunimmt. An dieser Stelle findet der Fehlerschätzer die größten Spannungsdifferenzen (Kap. 2.1.5). Es werden die relativen Fehler η_e in Prozent (Gl. (2.37)), durch Markierung der einzelnen Elemente entsprechend der angegebenen Skala, dargestellt. Je öfter das Netz adaptiv verfeinert wird, umso besser stimmt das berechnete Ergebnis mit der exakten Lösung überein. In diesem Beispiel wird die Verfeinerung bei einem relativen Fehler der Gesamtstruktur (Gl. (2.36)) von $\eta < \eta_{\text{Ziel}} = 5\%$ beendet, der hier nach fünf Iterationsschritten erreicht ist.
Ob ein Element verfeinert wird oder nicht, wird über Teilungsindikatoren $\tilde{\eta}_e$ berechnet, die sich aus dem ganzzahlig gerundeten Quotienten der relativen elementweisen Fehler η_e und der gewünschten relativen Fehlertoleranz η_{Ziel}

$$\tilde{\eta}_e = [\frac{\eta_e}{\eta_{\text{Ziel}}} + 0,5] \tag{4.2}$$

ergeben [35].
Die Verfeinerungsbedingung lautet dann

$$\begin{aligned} \tilde{\eta}_e > 1 \quad &: \quad \text{Element } e \text{ muss verfeinert werden} \\ \tilde{\eta}_e \leq 1 \quad &: \quad \text{Element } e \text{ muss nicht verfeinert werden.} \end{aligned} \tag{4.3}$$

Umgerechnet würde dies bedeuten, dass in den folgenden Abbildungen alle Elemente verfeinert werden, für die gilt: $\eta_e > 1,5 \cdot \eta_{\text{Ziel}} \underset{\text{hier}}{=} 7,5\%$.

Zu beachten ist, dass bei der adaptiven Netzverfeinerung in der Regel **inkompatible Knoten** entstehen. Dies sind Knoten, die zu einem finiten Element keine Verknüpfung besitzen. Im Allgemeinen ist dies in der FEM nicht erlaubt, da dadurch die wesentlichen inneren Randbedingungen verletzt werden, d.h. es kann zu Rissen innerhalb der Struktur kommen. Um dies zu verhindern, werden die Freiheitsgrade entsprechend der inkompatiblen Knoten beschränkt und mit kompatiblen Knoten durch „Multiple Point Constraints“ verknüpft [35].

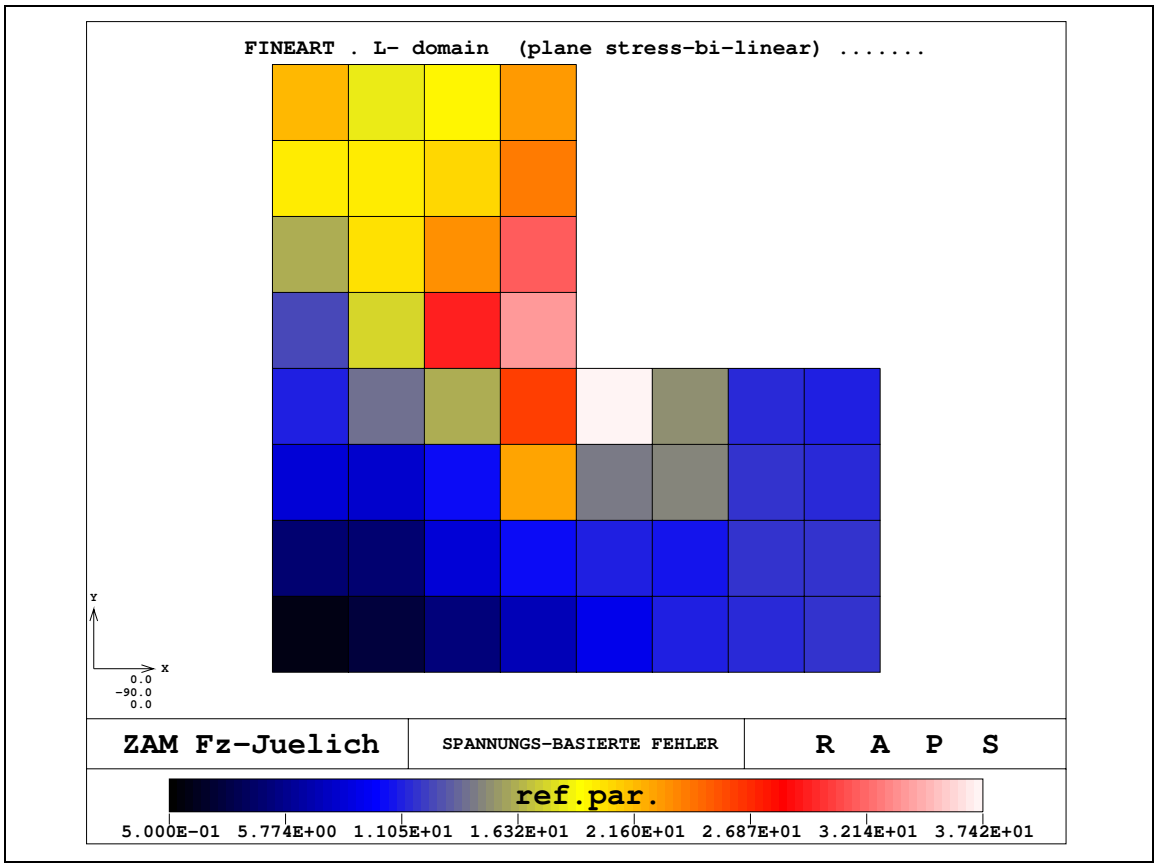


Abbildung 4.2: L-Domäne, Iterationsschritt 1 mit spannungs-basierten Fehlern

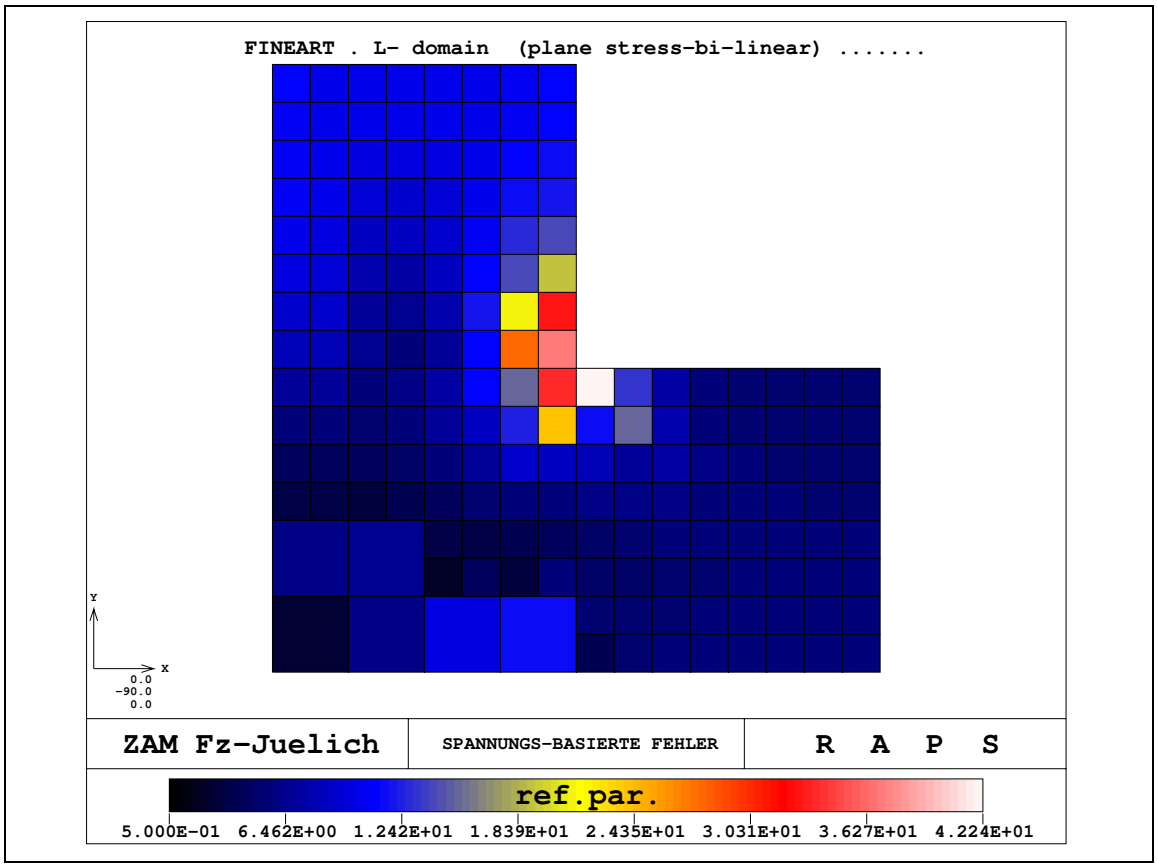


Abbildung 4.3: L-Domäne, Iterationsschritt 2 mit spannungs-basierten Fehlern

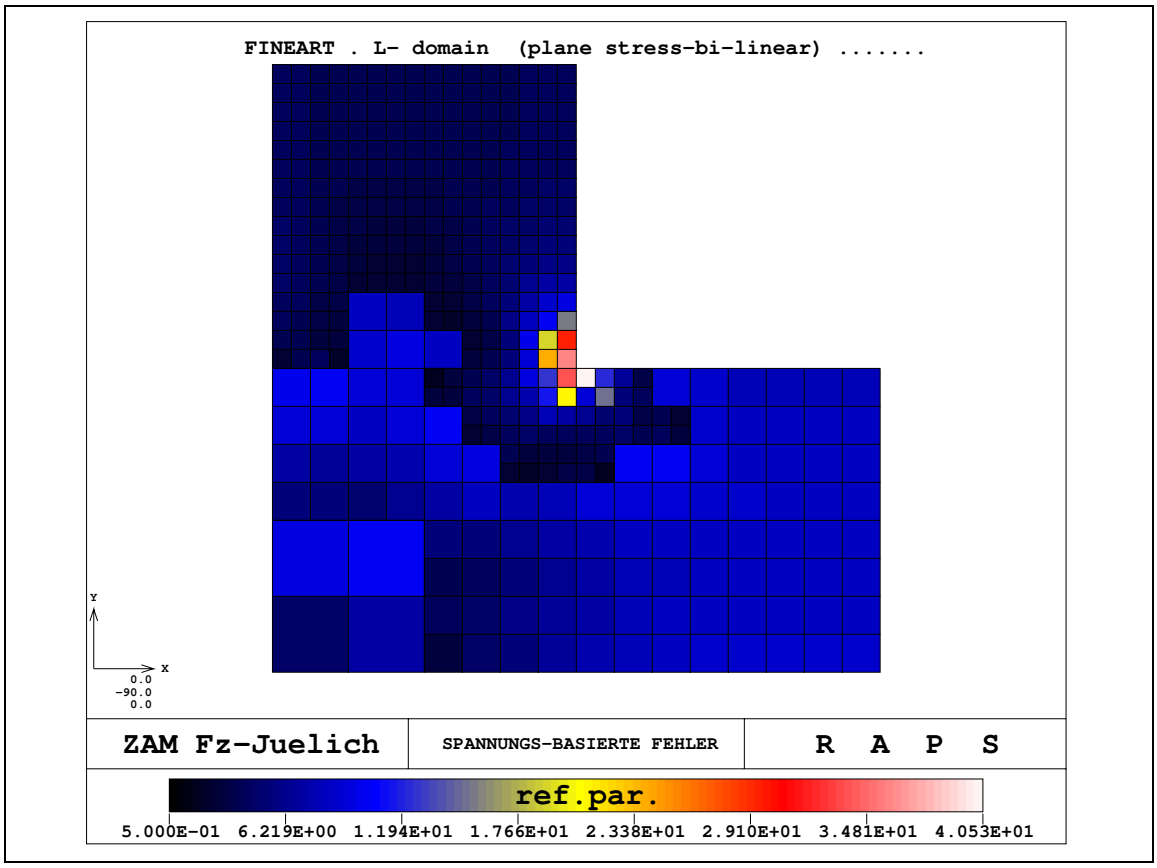


Abbildung 4.4: L-Domäne, Iterationsschritt 3 mit spannungs-basierten Fehlern

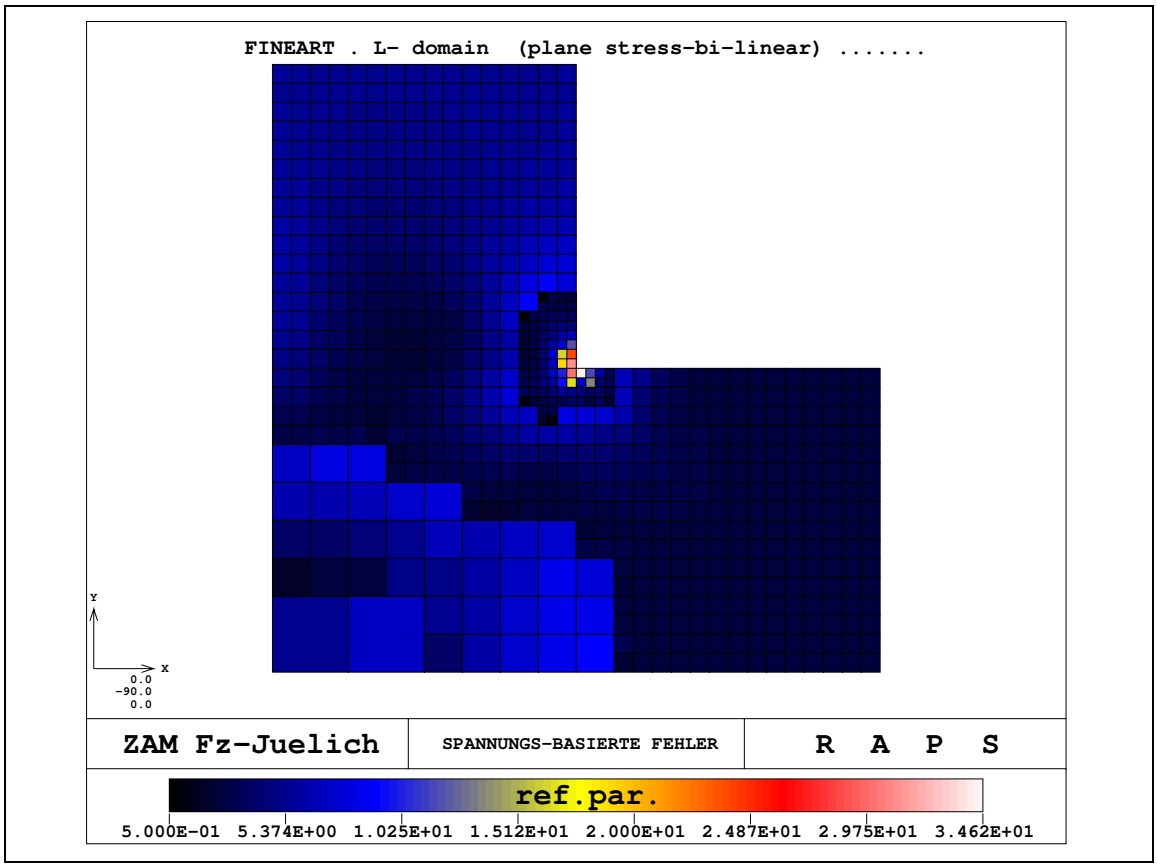


Abbildung 4.5: L-Domäne, Iterationsschritt 4 mit spannungs-basierten Fehlern

Im direkten Vergleich der Ergebnisse zwischen Iterationsschritt 0 und Iterationsschritt 5 sind bei den Verschiebungen mit bloßem Auge keine Unterschiede festzustellen (Abb. 4.6 und 4.7). Für diese Betrachtungen ist eine Netzverfeinerung nur dann notwendig, wenn präzise Berechnungen erforderlich sind.

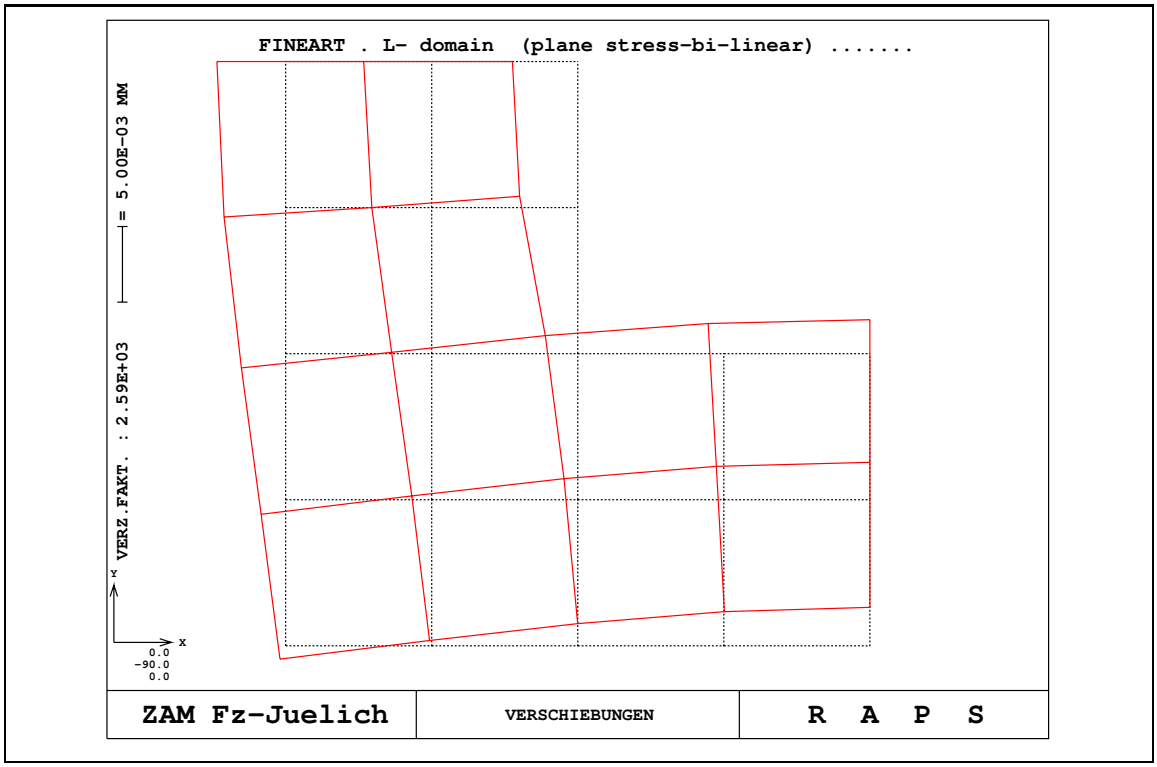


Abbildung 4.6: L-Domäne, Iterationsschritt 0 mit Verschiebungen

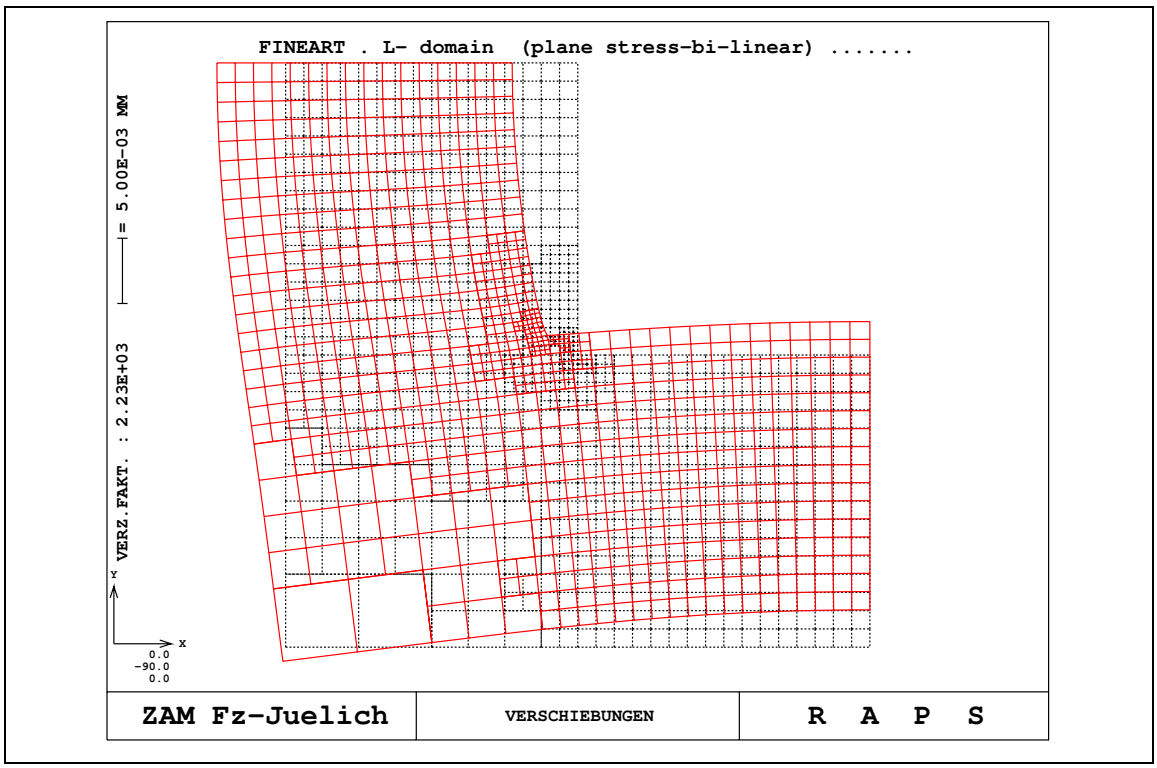


Abbildung 4.7: L-Domäne, Iterationsschritt 5 mit Verschiebungen

Ganz anders ist es bei den Spannungen. In Abbildung 4.8 bis 4.10 sind die jeweiligen von Mises Spannungen in Newton pro Quadratmillimeter dargestellt. Vor der Netzverfeinerung sieht es so aus, als sei die Spannung in Knoten 14, rechts neben der Kerbe am größten (Abb. 4.8). Beide, an die Kerbe grenzenden Seiten, bestehend aus Knoten 13, 14, 15, 18, 21 und zusätzlich Knoten 19 scheinen unter großen Spannungen zu stehen. Da jedoch auch starke Spannungsdifferenzen in den Elementen dieser Knoten zu erkennen sind, deutet dies auf eine verfälschte Darstellung.
Nach fünf iterativen Netzverfeinerungsschritten sieht das Ergebnis ganz anders aus. Dieses Modell kommt der Realität schon sehr nah (Abb. 4.9). In der Vergrößerung (Abb. 4.10) ist deutlich zu erkennen, dass die größten Spannungen im Bereich der Kerbe zu finden sind. Präzise ausgedrückt befindet sich die maximale Spannung in der Abbildung jedoch in dem Knoten rechts neben der Kerbe. Dies liegt daran, dass die Spannung σ_{nd} dieses Knotens einen höheren Spannungsbeitrag durch benachbarte Gauß-Punkte erhält, als der an der Kerbe (Kap. 2.1.5). Dadurch wird der Mittelwert insgesamt größer.
Des Weiteren ist zu erkennen, dass die berechnete Maximalspannung bei dem verfeinerten Netz mehr als doppelt so hoch ist, wie bei dem Ausgangsnetz, d.h auch die Absolutwerte weichen bei einem groben Modell oft stark von den wirklichen Größen ab. Die Erklärung liegt hier wieder in den höheren Spannungsbeiträgen durch benachbarte Gauß-Punkte. Bei feinen Netzen befinden sich die benachbarten Gauß-Punkte der Maximalstellen in einer näheren Umgebung als bei einem groben Netz, so dass bei diesen dann höhere Spannungen berechnet werden.

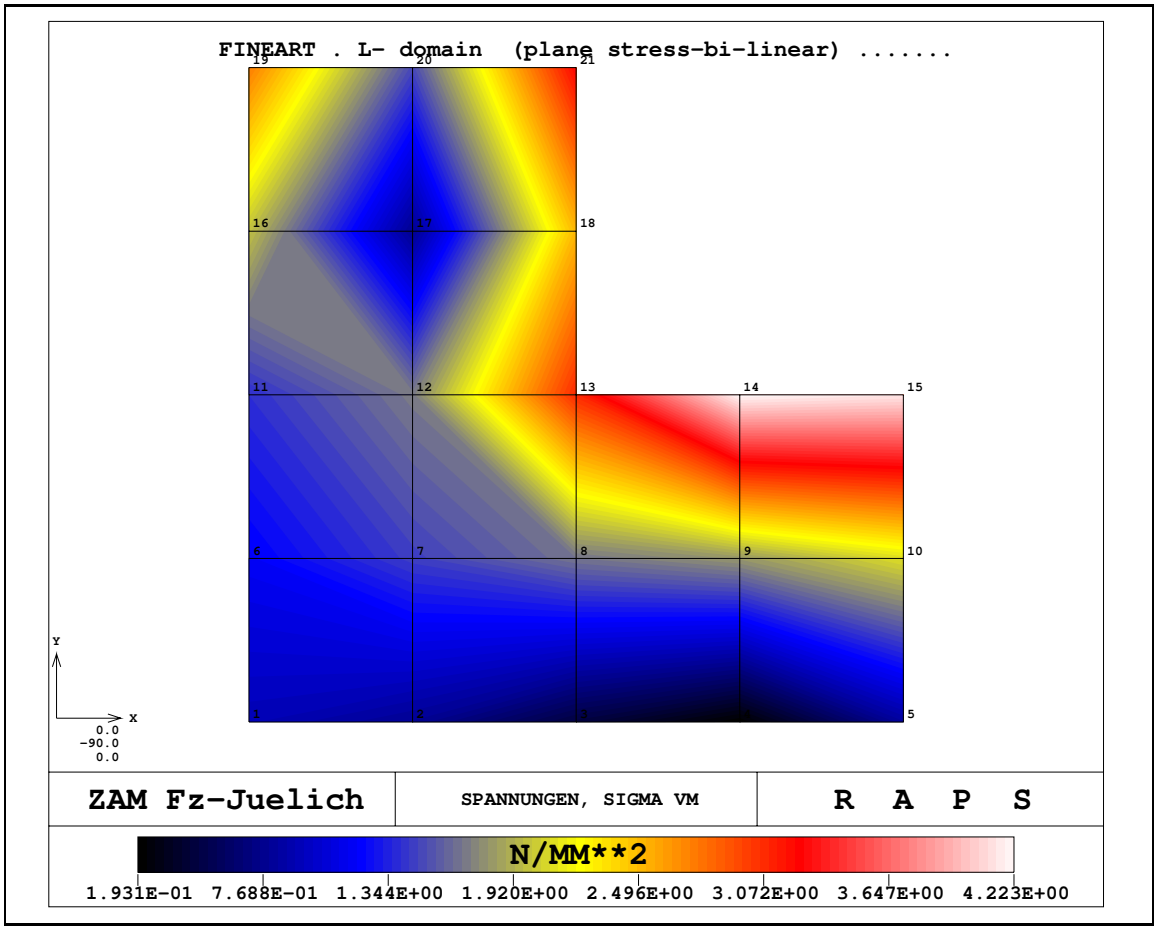


Abbildung 4.8: L-Domäne, Iterationsschritt 0 mit von Mises Spannungen und Knotennummern

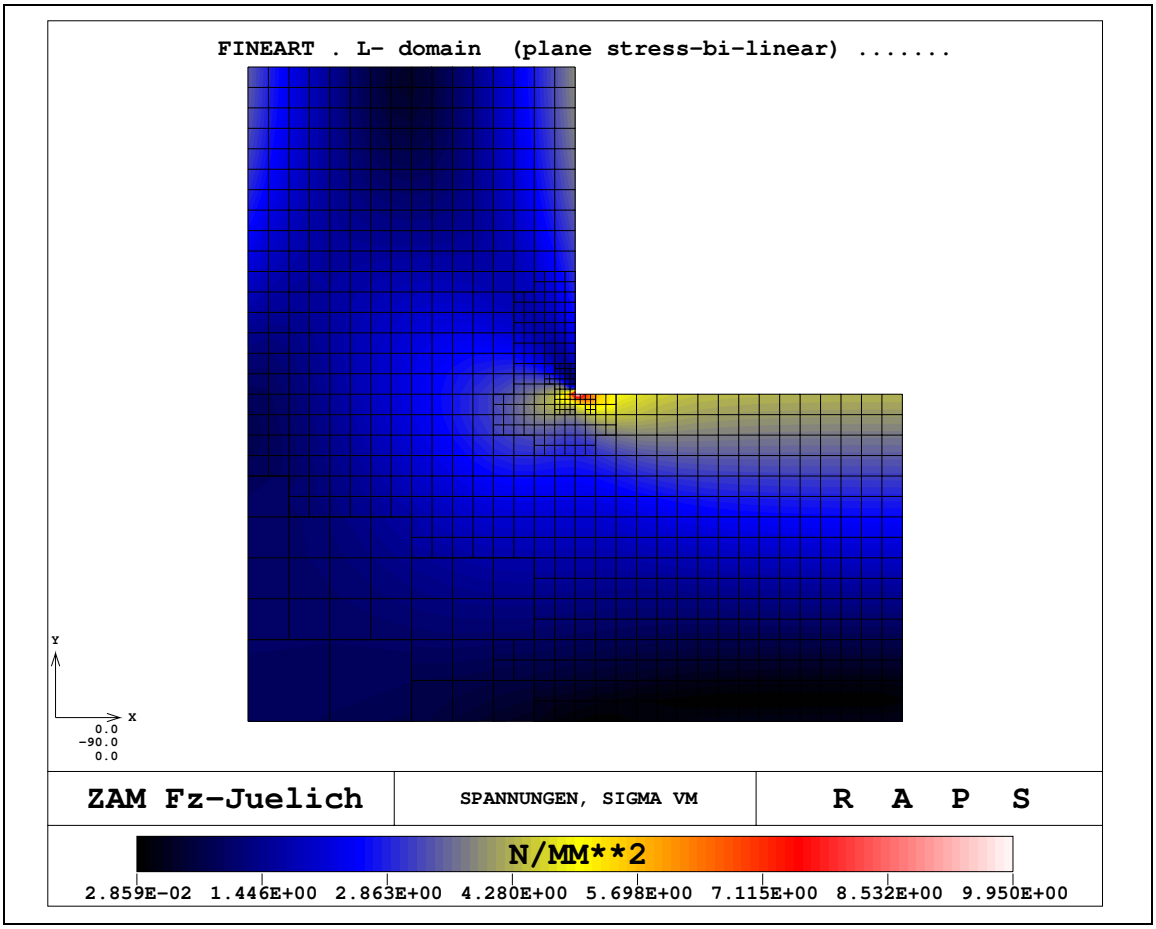


Abbildung 4.9: L-Domäne, Iterationsschritt 5 mit von Mises Spannungen

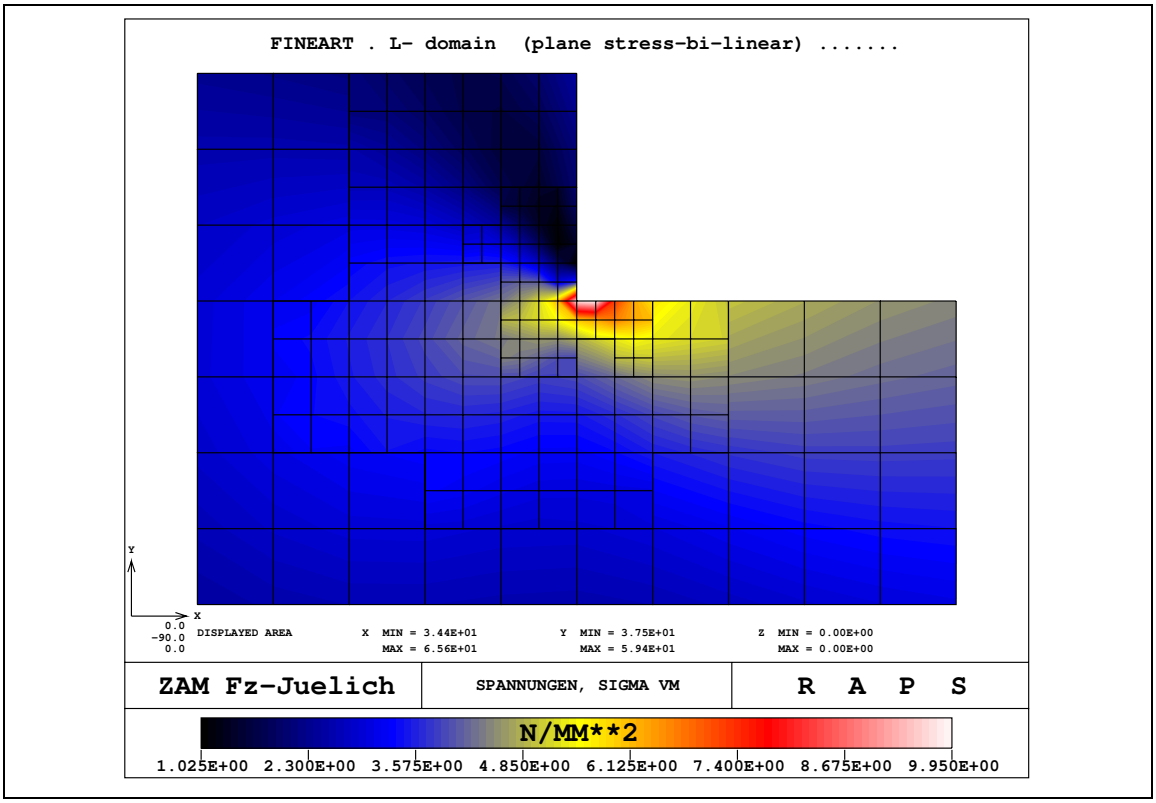


Abbildung 4.10: L-Domäne, Iterationsschritt 5 mit von Mises Spannungen, vergrößert

Um die Netzverfeinerung anhand der Spannungen zu verdeutlichen, werden in Abbildung 4.11 die von Mises Spannungen der L-Domäne beim nullten Iterationsschritt innerhalb der jeweiligen Elemente gezeigt. Die Rechnung wurde hier mit dem kommerziellen FEM-Programm PERMAS [36] durchgeführt, da von FINEART keine elementweisen Spannungen ausgegeben werden. Zur Datenkonvertierung wurden die Programme Perfi/Fiper genutzt, die von der Forschungszentrum Jülich GmbH entwickelt wurden und an deren Ergänzungen der Autor beteiligt war. Insbesondere in dem Knoten an der Kerbe kommt es zu starken Spannungsdifferenzen zwischen den benachbarten Gauß-Punkten dieses Knotens, so dass vor allem an dieser Stelle das Netz weiter verfeinert werden muss.

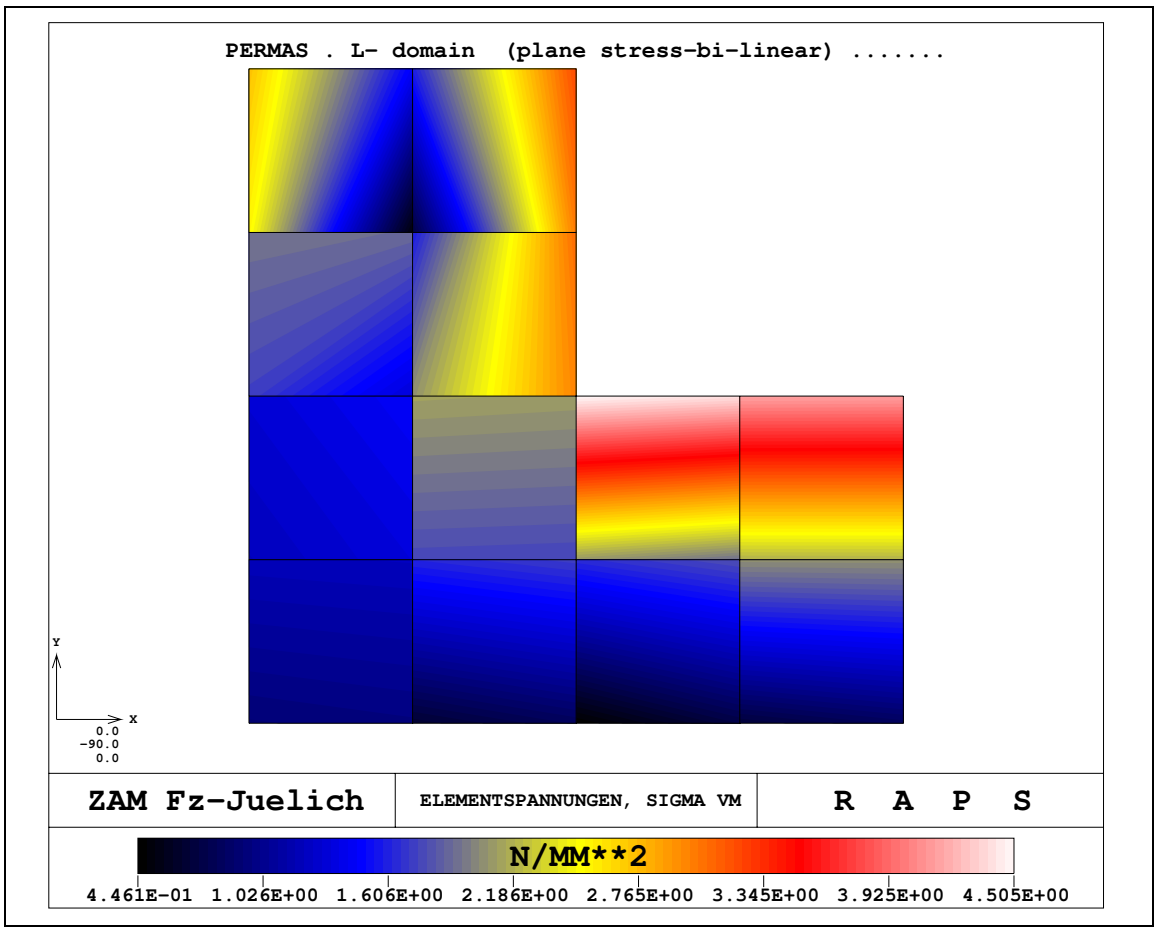


Abbildung 4.11: L-Domäne, Iterationsschritt 0 mit von Mises Spannungen (elementweise)

4.1.2 Weitere verwendete Strukturen

Da die aktuelle Version von FINEART noch in Entwicklung ist, stehen nur wenige Beispiele zur Verfügung, die für eine Untersuchung der Leistungsfähigkeit des neuen Lösungsmoduls genutzt werden können. So soll es in Zukunft auch möglich sein, dreidimensionale Strukturen zu berechnen. Dafür werden zurzeit insbesondere die Module für die Fehlerabschätzung und der adaptiven Netzverfeinerung überarbeitet. Gerade bei dreidimensionalen Objekten würde sich der Einsatz eines schnellen parallelen Gleichungslösers lohnen, da der Rechenaufwand dann um ein Vielfaches steigt.
Um trotzdem mit unterschiedlich komplexen Problemen rechnen zu können, wird für die folgenden Tests die L-Domäne mit einem feinen Ausgangsnetz benutzt. Zur Bestimmung der Elementzahl wird ein Bezeichner N eingeführt. N^2 ist dann die Anzahl der Elemente pro Quadrat. Für die bisher verwendete L-Domäne ist somit $N = 2$, die aus $3 \cdot 2^2 = 12$ Elementen besteht. Außerdem wird die

L-Domäne 100 mit $3 \cdot 100^2 = 30000$ Elementen eingesetzt.

Als weitere Struktur wird das zweidimensionale Modell eines Schraubenschlüsselsteils genutzt (Abb. 4.12). Es besteht zu Beginn aus 1038 verzerrten Vierecks-Elementen. Die Materialdaten und Lastgrößen entsprechen dabei denen der L-Domäne (Gl. (4.1)). Der untere Teil des Schlüssels wird sowohl in x- als auch in y-Richtung gefesselt. An den inneren Flächen links und rechts des Kopfes greifen Druckkräfte an, wodurch vereinfacht die Drehung an einer Schraube simuliert werden soll. Dieses Modell wurde mit dem Netzgenerator Quad-Gen [37] mit optimierter Knotennummerierung erstellt. Bei den L-Domänen hingegen werden die Knoten ohne Optimierung der Reihe nach durchnummeriert. Diese Modelle wurden mit einem selbst entwickelten Programm erzeugt.

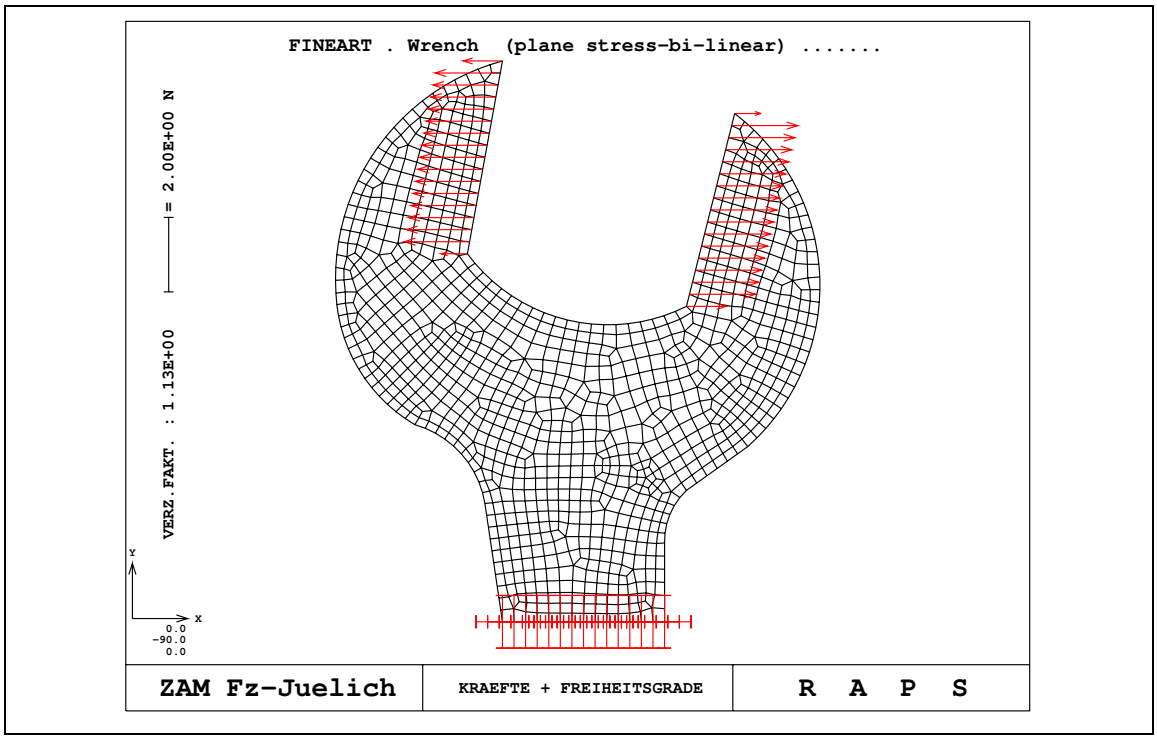


Abbildung 4.12: Schraubenschlüssel, Iterationsschritt 0 mit Lagerbeschreibung

Wie bei der L-Domäne sind die Ergebnisse vor einer Netzverfeinerung leicht verfälscht (Abb. 4.13). Da das Ausgangsnetz jedoch schon relativ fein ist, ist der Effekt wesentlich geringer. Nach vier Netzverfeinerungen sind in der Komplettansicht keine erhöhten Spannungen zu erkennen (Abb. 4.14), die jedoch in der Vergrößerung an den Kerben sichtbar werden (Abb. 4.15). Da der Kopf des Schraubenschlüssels relativ symmetrisch ist, sind die Spannungen an der rechten Seite ähnlich den an der linken Seite dargestellten. Dieses Beispiel demonstriert, dass grafische Ergebnisauswertungen auch mit einem genügend feinen Berechnungsnetz mit Vorsicht durchzuführen sind.

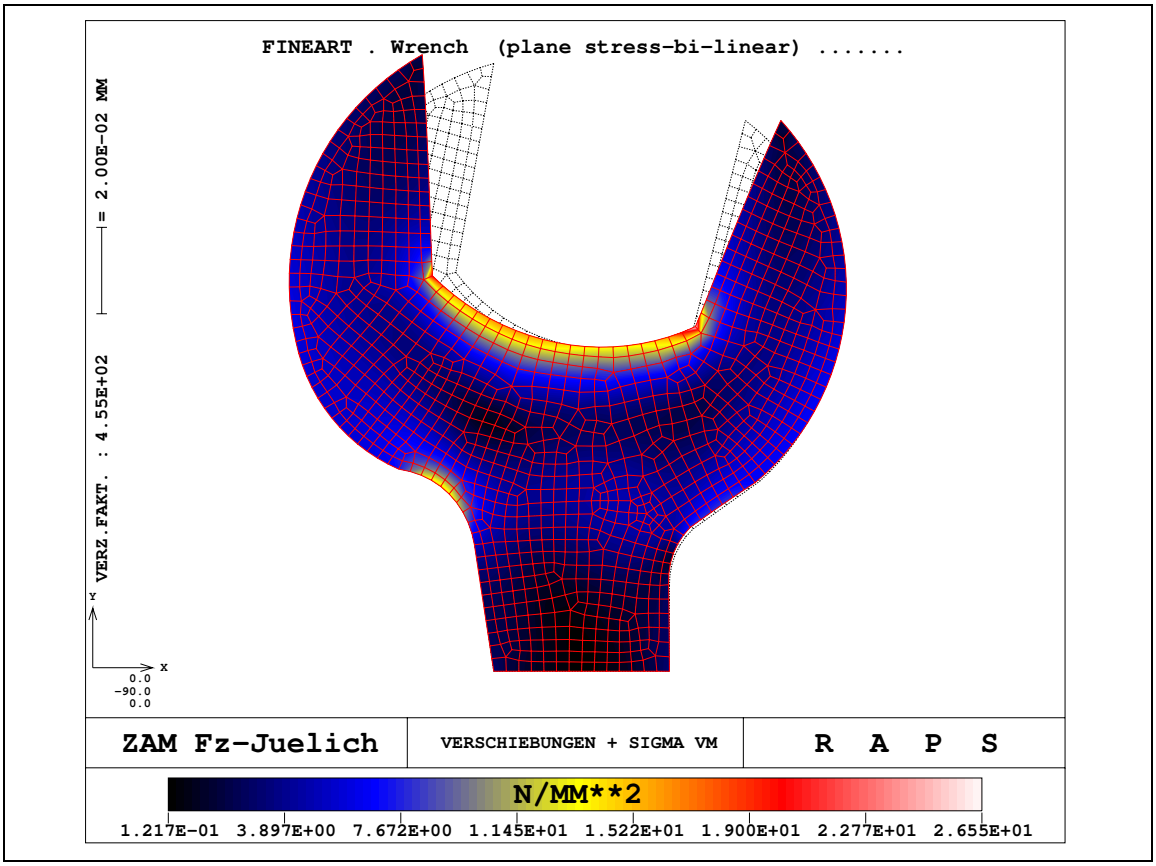


Abbildung 4.13: Schraubenschlüssel, Iterationsschritt 0 mit Verschiebungen und von Mises Spannungen

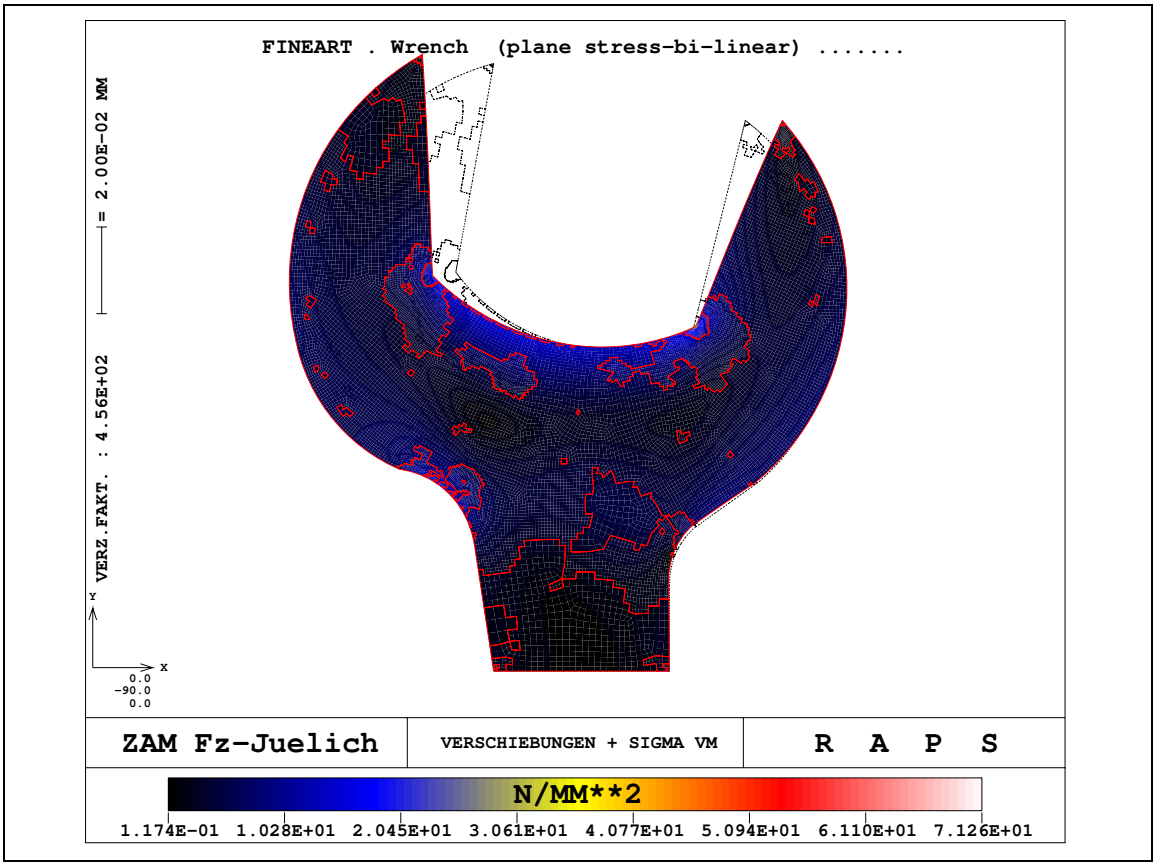
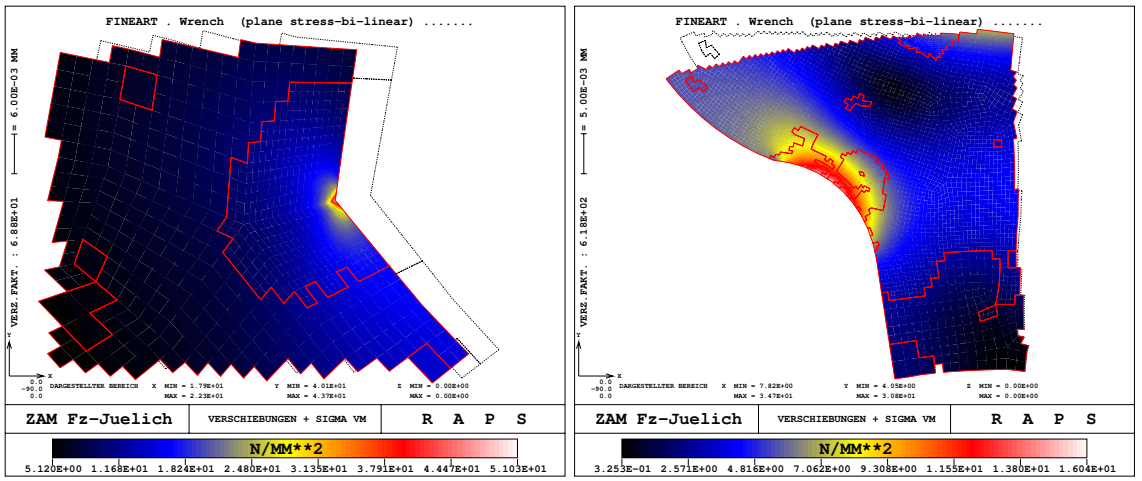


Abbildung 4.14: Schraubenschlüssel (Konturplot), Iterationsschritt 4 mit Verschiebungen und von Mises Spannungen



(a) Kerbe links innen

(b) Kerbe links außen

Abbildung 4.15: Schraubenschlüssel (Konturplot), Iterationsschritt 4 mit Verschiebungen und von Mises Spannungen, vergrößert

4.1.3 Bearbeitungszeit der einzelnen FINEART-Module und Änderung von Dimension und Bandbreite der Steifigkeitsmatrix bei jeder Iteration

In Tabelle 4.1 sind die benötigten Bearbeitungszeiten für die einzelnen FINEART-Module dargestellt, wobei die Zeit für die Fehlerabschätzung so gering ist, dass sie nicht mit angegeben wird. Es wird deutlich, dass sich der Einsatz eines parallelen Gleichungslösers für kleine Probleme (L-Domäne 2) nicht lohnt, da die Rechenzeiten selbst mit einem seriellen Löser nur einen Bruchteil einer Sekunde betragen. Zudem benötigen die anderen Module mehr oder mindestens etwa die gleiche Zeit. Selbst wenn ein paralleler Löser eingesetzt würde, würden sich die Rechenzeiten nicht verringern, da die Datenkommunikation dann mehr Zeit in Anspruch nimmt, als die Rechnung selbst, so dass dadurch ein zusätzlicher Aufand entsteht. Je feiner das Ausgangsnetz (L-Domäne 100) bzw. je komplexer die Modellgeometrie (Schraubenschlüssel) ist, umso mehr Zeit benötigt das Lösungsmodul, auch im Vergleich zu den anderen Modulen. Nur in diesem Fall würde sich der Einsatz eines parallelen Lösers lohnen. Nur das Modul der adaptiven Netzverfeinerung benötigt in der aktuellen Version mehr Zeit. Es ist jedoch geplant auch dieses Modul zu parallelisieren.

FEM-Problem	Iterationsschritt	0	1	2	3	4	5
L-Domäne 2	Berechnung Elementsteifigkeit	0,02	0,04	0,11	0,22	0,38	0,44
	Assemblierung globale Matrix	<0,01	<0,01	0,01	0,05	0,10	0,12
Fehlertoleranz	Lösen LGS (FINEART-Modul)	0,01	0,03	0,05	0,14	0,36	0,43
5 %	Adaptive Netzverfeinerung	0,03	0,04	0,13	0,25	0,30	-
L-Domäne 100	Berechnung Elementsteifigkeit	62,5	73,0	78,3	-	-	-
	Assemblierung globale Matrix	8,4	22,3	27,5	-	-	-
Fehlertoleranz	Lösen LGS (FINEART-Modul)	139	166	177	-	-	-
1 %	Adaptive Netzverfeinerung	456	473	-	-	-	-
Schraubenschl.	Berechnung Elementsteifigkeit	0,53	2,29	9,04	18,9	28,5	-
	Assemblierung globale Matrix	0,15	0,59	3,08	6,41	10,1	-
Fehlertoleranz	Lösen LGS (FINEART-Modul)	0,98	8,20	35,6	73,6	110	-
2 %	Adaptive Netzverfeinerung	6,23	46,2	127	207	-	-

Tabelle 4.1: Benötigte Zeit der einzelnen FINEART-Module bei jedem Iterationsschritt in Sekunden

Bei einem groben Ausgangsnetz (L-Domäne 2) wird zunächst meist, u.U. auch mehrfach, eine homogene Netzverfeinerung durchgeführt. Erst danach wird erkannt, an welchen Stellen adaptiv verfeinert werden kann, das jedoch von der gewählten Fehlertoleranz, die erreicht werden soll, abhängig ist (Gl. (4.2) und (4.3)). Dadurch erhöht sich die Matrixdimension relativ stark (Tab. 4.2). Bei einem feinen Ausgangsnetz (L-Domäne 100) hingegen wird nur adaptiv an den empfindlichen Stellen verfeinert, so dass sich die Matrixdimension nur relativ wenig erhöht. Die relative Bandbreite erhöht sich um ein Vielfaches, entgegengesetzt zur L-Domäne 2, bei der die relative Bandbreite verhältnismäßig konstant bleibt, bzw. sich insgesamt verringert. Daraus lässt sich vermuten, dass die Umnummerierung der Knoten nach jeder Iteration, zumindest für sehr feine Ausgangsnetze in Bezug auf Bandbreitenminimierung nicht besonders effektiv ist. Bei einer etwas komplexeren Struktur (Schraubenschlüssel) kommt es, abgesehen von dem zweiten Iterationsschritt, wie bei der L-Domäne 2 zu einer Verringerung der relativen Bandbreite. Um beurteilen zu können, wann eine Knotennummerierung eine geringe Bandbreite bringt und wann nicht, müssten genauere Untersuchungen gemacht werden, was hier wegen der fehlenden Beispiele nicht möglich ist. Es ist aber deutlich erkennbar, dass die Knotennummerierung insgesamt verbesserungswürdig ist, falls eine geringe Bandbreite von Bedeutung ist.

FEM-Problem	Iterationsschritt	0	1	2	3	4	5
L-Domäne 2	Dimension	36	120	384	838	1446	1644
	relative Vergrößerung [%]	-	233	220	118	73	14
Fehlertoleranz	Bandbreite	12	25	49	145	197	215
5 %	relative Bandbreite [%]	33,33	20,83	12,76	17,30	13,62	12,92
L-Domäne 100	Dimension	60600	65842	68312	-	-	-
	relative Vergrößerung [%]	-	8,7	3,8	-	-	-
Fehlertoleranz	Bandbreite	404	1149	1377	-	-	-
1 %	relative Bandbreite [%]	0,67	1,75	2,02	-	-	-
Schraubenschl.	Dimension	2220	7432	18762	29862	37484	-
	relative Vergrößerung [%]	-	235	152	59	26	-
Fehlertoleranz	Bandbreite	175	251	703	875	1067	-
2 %	relative Bandbreite [%]	7,88	3,38	3,75	2,93	2,85	-

Tabelle 4.2: Vergrößerung der Matrixdimension und deren Bandbreite in FINEART mit jedem Iterationsschritt

4.2 Messungen zur Leistungsfähigkeit des neuen Lösungsmoduls

Die folgenden Messungen werden mit der etwas komplexeren Schraubenschlüssel-Struktur und einem feinen Ausgangsnetz der L-Domäne zu verschiedenen Iterationsschritten der Netzverfeinerung durchgeführt. Nur so ist es sinnvoll, den parallelen Gleichungslöser einzusetzen (Kap. 4.1.3), da dann das FEM-Netz aus vielen Elementen besteht. Für den Gleichungslöser macht es keinen Unterschied, ob es sich um eine komplexe Geometrie oder ein feines Netz handelt, da seine Effizienz nur von der Dimension und der Bandbreite der Steifigkeitsmatrix abhängt.
Die in den Diagrammen angegebenen Zeiten sind die Gesamtzeiten des Lösermoduls (Ein-/Ausgabe, Berechnung des LGS, Ergebnisüberprüfung). Da die Zeit für die Berechnung des LGS bei dieser Problemgröße meist über 90 Prozent beträgt, können die anderen Größen jedoch so gut wie vernachlässigt werden und werden deshalb nicht weiter untersucht. Nur bei der Nutzung nur eines Prozesses ist der Anteil der Zeit für die Ergebnisüberprüfung relativ hoch, so dass die Zeit für die Berechnung des LGS dann nur mindestens 70 Prozent beträgt. Dies liegt daran, dass die Rechenzeit für die Ergebnisüberprüfung kontinuierlich abnimmt, je mehr Prozesse genutzt werden. Entgegengesetzt kommt es bei der Berechnung des LGS bei zwei Prozessen zu einer Zunahme der Rechenzeit, wodurch der relative Anteil der Ergebnisüberprüfung abnimmt.

Es wird der serielle FINEART Löser für dünn besetzte Matrizen mit Löser C und der parallele ScaLAPACK Bandlöser wieder mit Löser B bezeichnet. Der parallele Löser für dicht besetzte Matrizen wird in diesen Auswertungen nicht berücksichtigt, da die Matrizen eine relativ geringe Bandbreite haben, die im Bereich von unter zehn Prozent der Matrixdimension liegen. In Kapitel 3.5.2 wurde gezeigt, dass dieser Löser im Allgemeinen für Bandbreiten dieser Größenordnung ineffizienter ist, als der Löser für Bandmatrizen (Abb. 3.16). Zudem kann dieser Löser bei sehr komplexen Problemen gar nicht genutzt werden, da der vorhandene Speicher für diese Matrixgrößen nicht ausreicht.

Der Kurvenverlauf von Löser B der L-Domäne 100 beim nullten Iterationsschritt (Abb. 4.16) entspricht etwa dem zuvor getesteten Beispiel aus Kapitel 3.5.2 mit einer großen Matrix und kleiner Bandbreite (Abb. 3.16). Es werden somit die vorhergehenden Beobachtungen bestätigt. Insbesondere zeigt sich wieder, dass es bei sehr großen Datenmengen bei 4 Prozessen zu einer Zunahme der Rechenzeit kommt. Der Grund dafür ist, dass alle 4 Prozessoren eines SMP-Knotens genutzt werden und es so häufig zu Kollisionen bei der Datenkommunikation kommt. Der selbe, allerdings nicht so starke Effekt kann hier sogar bei 8 Prozessen festgestellt werden. In diesem Fall werden alle 4 Prozessoren zweier SMP-Knoten verwendet. Bei 12 Prozessen, wenn alle Prozessoren dreier SMP-Knoten genutzt werden, ist eine entsprechende Zunahme der Rechenzeit wiederum kaum sichtbar.
Die benötigte Rechenzeit des parallelen Algorithmus mit der maximalen Anzahl an Prozessen liegt nur geringfügig unter der mit einem Prozess. Der Grund für die geringe Leistungssteigerung ist wiederum die zurzeit installierte MPI-Version bzw. das, im Vergleich zur CRAY T3E, relativ langsame Verbindungsnetzwerk des ZAMpano. Die Zunahme der Rechenzeit von einem nach zwei Prozessen ist mit Faktor 5 etwas niedriger, als bei dem Beispiel aus dem vorigen Kapitel, da auch die Matrix etwas kleiner ist. Die minimale Gesamtzeit von Löser B beträgt nur etwa 35 % der Zeit von Löser C.

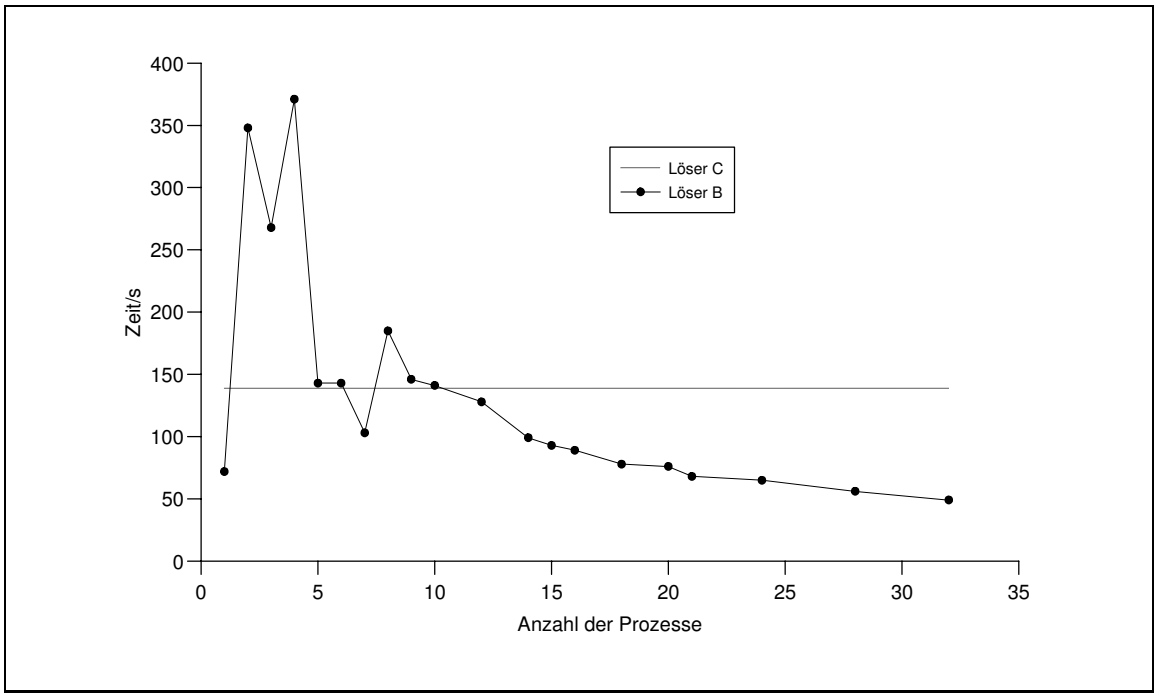


Abbildung 4.16: L-Domäne 100, Iterationsschritt 0, Matrix der Dimension 60600 mit Bandbreite 404

Bei der L-Domäne 100 nach dem ersten Iterationsschritt (Abb. 4.17) sind die Kurvenverläufe von Löser B ebenfalls wieder ähnlich. Jedoch ist die minimale Rechenzeit von Löser B etwa doppelt so hoch, wie die von Löser C. Dies liegt an der eher ungünstigen Neunummerierung der Knoten in diesem Beispiel für Löser B, da es dann zu einer relativen Vervielfachung der Bandbreite im Ver-

gleich zur Matrixdimension kommt. So wird Löser B im Zeitverhalten mit jedem Iterationsschritt bei diesem Beispiel deutlich schlechter.

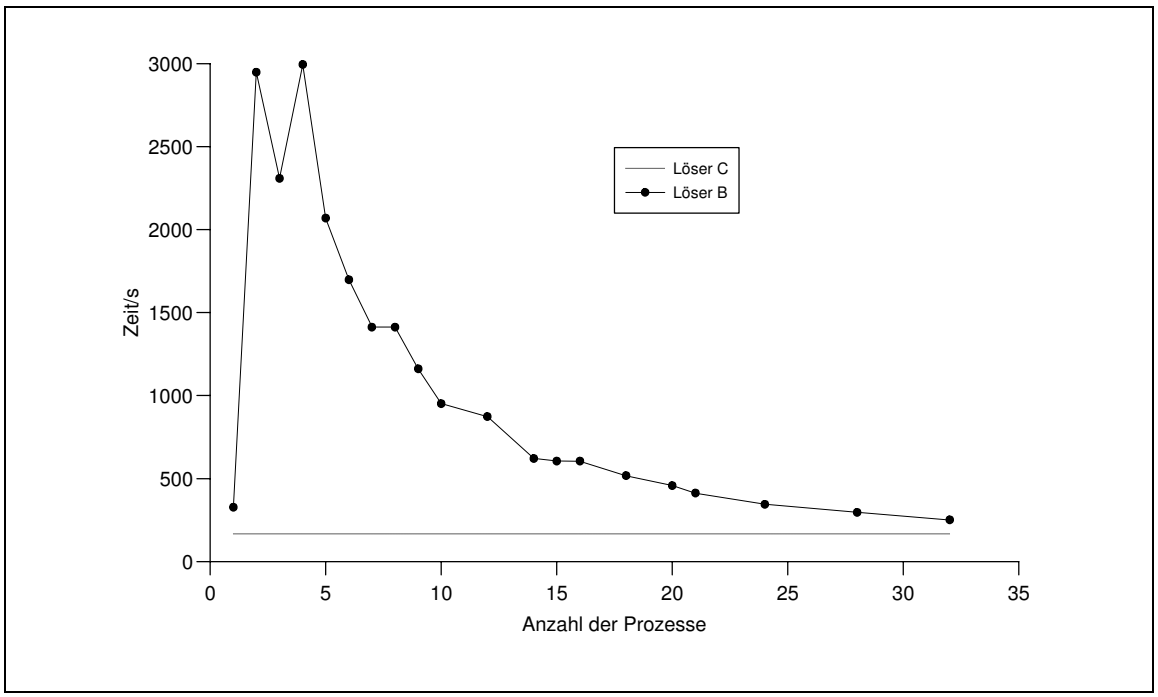


Abbildung 4.17: L-Domäne 100, Iterationsschritt 1, Matrix der Dimension 65842 mit Bandbreite 1149

Bei dem Schraubenschlüssel kommt es, trotz der höheren Komplexität der Geometrie wegen dem eher groben Ausgangsnetz zu einer kleineren Matrix, als bei der L-Domäne 100. Deshalb entspricht der Kurvenverlauf von Löser B nach dem ersten Iterationsschritt (Abb. 4.18) etwa dem der kleinen Matrix mit einer kleinen Bandbreite, die im vorigen Kapitel getestet wurde (Abb. 3.14). Die minimale Rechenzeit ist dabei etwa halb so groß, wie die von Löser C, was auf eine, für Löser B günstige Neunummerierung deutet.

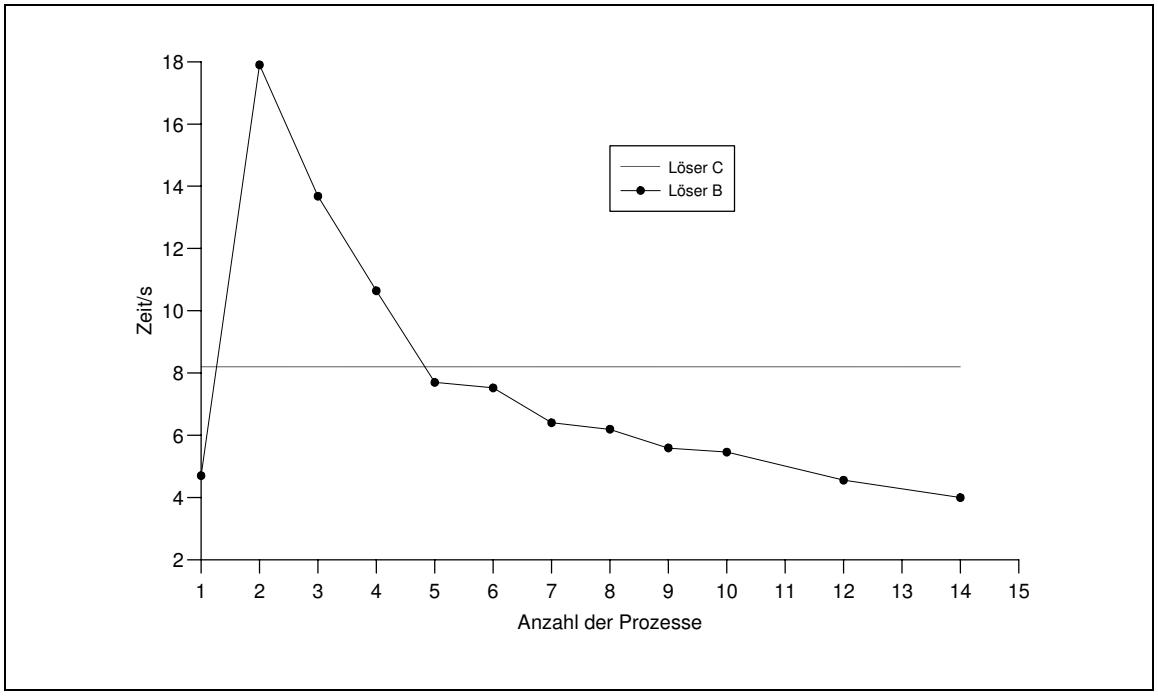


Abbildung 4.18: Schraubenschlüssel, Iterationsschritt 1, Matrix der Dimension 7432 mit Bandbreite 251

Ein ähnlicher Kurvenverlauf für Löser B lässt sich ebenfalls nach dem zweiten Iterationsschritt feststellen. Dabei liegt die minimale Rechenzeit diesmal etwas über der von Löser C. Daraus lässt sich auf eine eher ungeeignete Neunummerierung für diese Iteration schließen. Tabelle 4.2 belegt, dass die relative Bandbreite hier zugenommen hat. Bei allen anderen Iterationen dieses Beispiels nimmt sie ab.

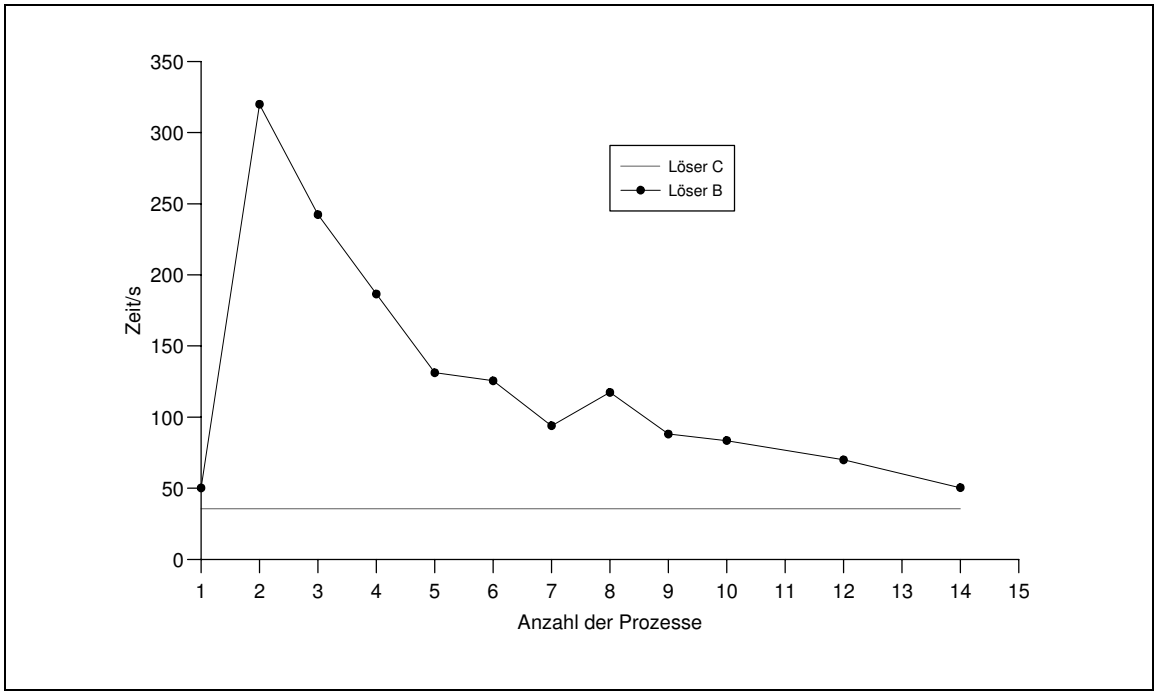


Abbildung 4.19: Schraubenschlüssel, Iterationsschritt 2, Matrix der Dimension 18762 mit Bandbreite 703

Da auf dem ZAMpano zurzeit umfangreiche Tests mit einer MPI-Version durchgeführt werden, die die Ausnutzung von gemeinsamen Speicherbereichen unterstützt, können keine weiteren Messungen durchgeführt werden. Es werden jedoch auch so alle notwendigen Ergebnisse dargestellt. Die Tests sind bisher sehr vielversprechend und es sieht so aus, als würde die alte MPI-Version bald ersetzt werden.

Kapitel 5

Zusammenfassung

Das Ziel dieser Arbeit war eine Effizienzsteigerung des Lösungsmoduls von FINEART durch Implementierung eines parallelen Gleichungslösers. In Kapitel 2.1.2 wurde gezeigt, dass die Steifigkeitsmatrizen der linearen Gleichungssysteme (LGS), die bei der FEM entstehen, symmetrisch, positiv definit und dünn besetzt sind. Dies führt, bei geeigneter Knotennummerierung zu einer Bandmatrix. Dem entsprechend wurden zwei verschiedene direkte Löser der parallelen Programmbibliothek ScaLAPACK untersucht, die LGS mit Matrizen dieser Eigenschaften berechnen können. Dabei handelt es sich um einen Löser für Bandmatrizen und einen für dicht besetzte Matrizen. Ein Löser für dünn besetzte Matrizen ist in ScaLAPACK nicht enthalten. Stattdessen wurde der Löser für dicht besetzte Matrizen für Vergleiche hinsichtlich der benötigten Rechenzeit und des Speicherplatzes benutzt. In Kapitel 2.2 wurde beschrieben, wie ein Problem allgemein parallel von mehreren Prozessen bearbeitet wird und in Kapitel 3 wurden diese Angaben für ScaLAPACK bzw. die beiden benutzten Löser spezifiziert.
Die Anwendung paralleler Lösungsverfahren lohnt sich nur für genügend große Aufgaben bzw. Datenmengen. Dies gilt ebenso für das Lösen linearer Gleichungssysteme, so dass nur entsprechende Beispiele untersucht wurden. Zunächst wurden von FINEART unabhängige Messungen auf dem SMP-Cluster ZAMpano bzw. auf dem massiv parallelen System CRAY T3E durchgeführt (Kap. 3.5). Die dabei verwendeten Matrizen sind größtenteils Beispiele aus Finite-Elemente-Berechnungen, die im Matrix Market veröffentlicht wurden. Wie zu erwarten ist, liefert der Löser für dicht besetzte Matrizen bei eher kleinen Bandbreiten wesentlich höhere Rechenzeiten, als der Löser für Bandmatrizen. In den getesteten Beispielen ist aber selbst mit relativ großer Bandbreite der Löser für Bandmatrizen noch besser geeignet. Bei großen Problemen mit hohen Matrixdimensionen ist es sogar so, dass der Löser für dicht besetzte Matrizen gar nicht mehr verwendet werden kann, da dieser mehr Speicherplatz benötigt, als der Löser für Bandmatrizen. Der Löser für Bandmatrizen ist diesem Löser demnach sowohl in Bezug auf Minimierung der Rechenzeit, als auch des benötigten Speicherplatzes bei diesen Beispielen zu bevorzugen. Die folgenden Untersuchungen wurden deshalb auf den Bandlöser beschränkt.
Es zeigte sich, dass dieser parallele Bandlöser im Hinblick auf Rechenzeitminimierung, zumindest auf dem ZAMpano und zum jetzigen Stand der Soft- bzw. Hardware, nur geringe oder gar keine Verbesserung bringt, da die geringsten Rechenzeiten meist bei serieller Ausführung des Lösers für Bandmatrizen erzielt wurden. Gute Rechenzeiten ergaben sich bei paralleler Ausführung nur bei großen Matrizen mit extrem kleinen Bandbreiten. Zudem sollte bei der Nutzung des parallelen Lösers auf dem ZAMpano beachtet werden, dass durch die zurzeit installierte MPI-Version geringere Prozesszahlen teilweise bessere Rechenzeiten liefern, als eine große Anzahl an Prozessen, da diese Version keine Unterstützung für gemeinsame Speicherbereiche bietet. Auf der CRAY T3E hingegen tritt dieser Effekt nicht auf, da dies ein System mit verteiltem Speicher ist. Ebenso werden auf dieser Maschine, insbesondere bei paralleler Ausführung, bessere Rechenzeiten erzielt, als auf dem ZAMpano, da die CRAY T3E ein schnelleres Verbindungsnetzwerk besitzt.

Schließlich wurde nach einer Beschreibung des FEM-Programms FINEART in Kapitel 4 belegt, dass sich der Einsatz eines parallelen Lösers in FINEART nur für komplexe Probleme eignet. Zudem wurde gezeigt, dass in diesen Fällen das Lösungsmodul, abgesehen von dem Modul der Netzverfeinerung, die meiste Zeit in Anspruch nimmt, so dass eine Erhöhung der Leistungsfähigkeit dieses Moduls sinnvoll ist. Außerdem wurde untersucht, wie sich Dimension und Bandbreite der Steifigkeitsmatrix mit jedem Verfeinerungsschritt ändern. Dabei wurde festgestellt, dass es oft zu einer Erhöhung der relativen Bandbreite kommt. Die Neunummerierung der Knoten ist deshalb teilweise verbesserungswürdig.
Anschließend wurde der ScaLAPACK Bandlöser mit dem bisher in FINEART implementierten seriellen Löser verglichen. Dieser basiert auf einem iterativen Verfahren für dünn besetzte Matrizen. Bei allen, mit FINEART getesteten Beispielen entstehen Steifigkeitsmatrizen, deren Parameter, d.h. Dimension und Bandbreite, ähnlich denen der Matrizen aus dem Matrix Market sind. Deshalb ist auch das Verhalten des Bandlösers mit FINEART, bis auf geringe Abweichungen, identisch dem mit vergleichbaren Matrizen aus dem Matrix Market. Dabei ist die minimale Rechenzeit des parallelen Lösers für Bandmatrizen teilweise, je nach Knotennummerierung, geringer als die des FINEART-Lösers.

Als Fazit lässt sich sagen, dass durch die Implementierung des ScaLAPACK-Bandlösers auf dem ZAMpano zum jetzigen Zeitpunkt das Ziel einer Effizienzsteigerung gegenüber dem bisherigen FINEART-Löser nur bedingt erreicht wurde. Je nach Problem und Netzverfeinerungsschritt führt FINEART eine mehr oder weniger effektive Knotennummerierung in Bezug auf Bandbreitenminimierung der Steifigkeitsmatrix durch. Für den FINEART-Löser spielt dies keine Rolle, da dieser unabhängig von der Bandbreite ist. Dies ist jedoch äußerst wichtig für den ScaLAPACK-Bandlöser, um geringe Rechenzeiten und Speicherplatzanforderungen erzielen zu können. So liefert der ScaLAPACK-Bandlöser bei manchen Beispielen bessere Ergebnisse, als der FINEART-Löser und umgekehrt. Dabei sind die Rechenzeiten des Bandlösers bei serieller Ausführung auf dem ZAMpano etwa identisch denen bei der parallelen Ausführung mit der maximalen Anzahl an Prozessen.
Um die Effizienz des parallelen Lösungsmoduls weiter zu steigern, gibt es die folgenden Möglichkeiten:
Als erstes sollte eine effizientere MPI-Version auf dem ZAMpano installiert werden, die die physikalische Prozessortopologie besser ausnutzt. Dazu werden derzeit vielversprechende Tests durchgeführt. Dies ist auf jeden Fall sinnvoll, da dieses Manko sich auch bei anderen parallelen Anwendungen bemerkbar macht. Auf Nutzerebene könnte die Neunummerierung der Knoten nach jeder Iteration verbessert werden, um geringere Bandbreiten zu erreichen. Eine andere Alternative wäre die Implementierung eines parallelen Lösers für dünn besetzte Matrizen, der wie der bisher in FINEART implementierte Löser, unabhängig von der Bandbreite arbeitet. Sind nach all diesen Maßnahmen weitere Verbesserungen erforderlich, bleibt nur der Wechsel auf einen anderen Parallelrechner oder die Verwendung eines schnelleren Verbindungsnetzwerkes auf dem ZAMpano.
Um die gesamte Rechenzeit von FINEART bei mehreren Verfeinerungsschritten zu verringern, sollte zusätzlich mindestens das Modul der adaptiven Netzverfeinerung verbessert werden, da gerade dieses die meiste Zeit bei der Problembearbeitung benötigt. Entsprechende Pläne existieren im Rahmen des Kooperationsprojekts mit dem Structural Engineering Research Centre, Madras bereits.

Anhang A

Details zur Implementierung des Lösungsmoduls

A.1 Aufbau des neuen Moduls

Das neue Modul zum Lösen des LGS ist, ebenfalls wie FINEART selbst, in mehrere Untermodule unterteilt. Dabei wird als Programmiersprache FORTRAN 90 benutzt. Das Hauptmodul *solver.f* ist, wie in Abbildung A.1 dargestellt, aufgebaut.

Dateioperationen werden nur von einem Hauptprozess ausgeführt (seriell). Dieser hat den größten Arbeitsaufwand zu bewältigen und bestimmt somit die, in den Messergebnissen ausgegebene, Wartezeit des Benutzers. Es werden MPI-Routinen genutzt, um die Daten an die anderen Prozesse zu senden, bzw. von den anderen Prozessen zu empfangen. Es wäre theoretisch möglich, auch den parallelen Lese- bzw. Schreibzugriff auf Dateien zu realisieren. MPI-2, ein neuerer Standard des „Message Passing Interface“ erlaubt den Einsatz dieses Verfahrens. Es ist jedoch noch nicht so verbreitet wie MPI-1. Zudem muss das Dateisystem des Rechners den parallelen Zugriff unterstützen, was nicht immer der Fall ist. Ein weiteres Problem ist, dass die bisherige Ein-/ Ausgabe nicht für einen parallelen Dateizugriff konzipiert ist.
Die einzelnen Elemente der Matrix bzw. RHS werden beim Lesen in einem zweidimensionales Feld gespeichert. Die erste Dimension bestimmt den Empfängerprozessor, die zweite die Nummer des Elementeintrags. Gespeichert wird eine Datenstruktur, bestehend aus Zeilen bzw. Spaltenposition des Elementeintrags in der Matrix des Empfängers sowie der Elementeintrag selbst. Die Bestimmung der Submatrizen der einzelnen Prozesse entsprechend den in Kapitel 3.3 beschriebenen Schemata wird so schon vor dem Versenden der Daten vorgenommen. Erst wenn der Einlesevorgang endgültig abgeschlossen wurde, werden die Daten versendet.
Zur Datenkommunikation werden nicht Befehle von BLACS, sondern von MPI benutzt. Dies ist ohne Probleme möglich, da keine zusätzlichen Kommunikationsvorgänge innerhalb einer Matrix oder eines Vektors programmiert wurden. Aus diesem Grund würde BLACS keine Effizienzsteigerung bringen und das bekanntere MPI kann ebenfalls verwendet werden. Das Prozessgitter muss jedoch trotzdem erstellt werden, insbesondere da BLACS von den Lösungsalgorithmen verwendet wird.
Die Ergebnisüberprüfung kann manuell über einen Parameter ein- oder ausgeschaltet werden, um Speicherplatz zu sparen. Da die ursprüngliche Matrix dafür benötigt wird, wäre so nahezu der doppelte Speicherplatz erforderlich. Die Überprüfung kann auch automatisch abgeschaltet werden, indem eine Obergrenze für den zur Verfügung stehenden Speicher für die Daten des Gleichungssystems angegeben wird. Wird mehr Speicher benötigt, als zulässig, wird keine Ergebnisüberprüfung durchgeführt.

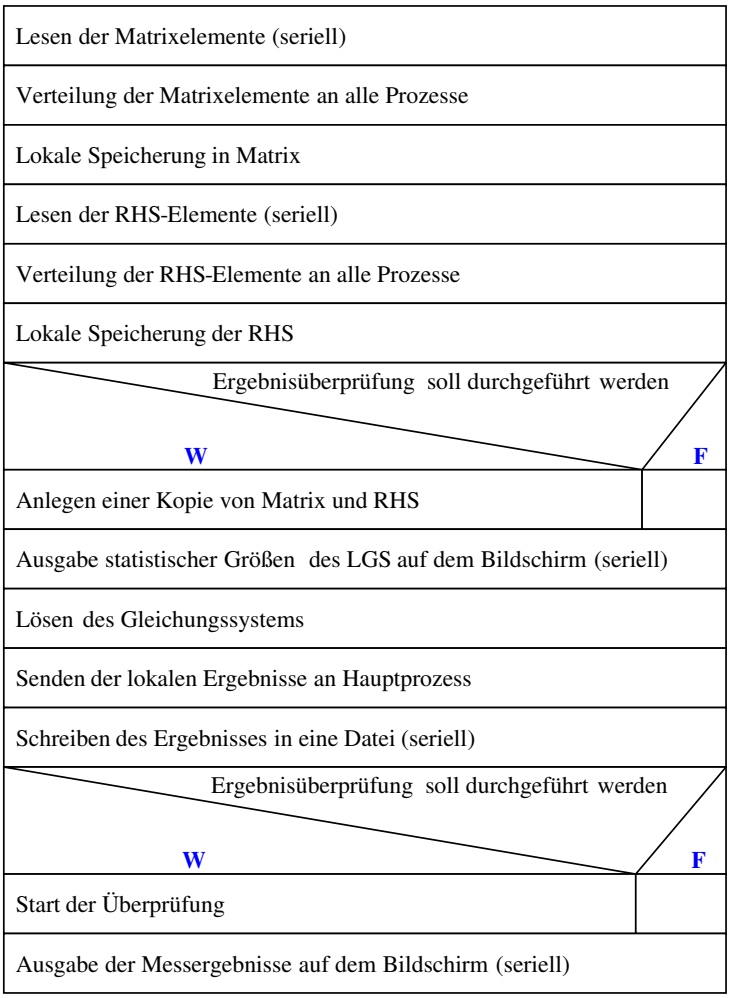


Abbildung A.1: Struktogramm des neuen Hauptmoduls

Neben dem Hauptmodul existiert noch ein Modul *modules.f*, das einige Variablendeklarationen und weniger bedeutungsvolle Funktionen enthält, die eingebunden werden müssen, sowie zwei Module die Funktionen für das Lesen bzw. Schreiben der Daten (I/O). Im ersten Modul *inout.f* werden FINEART-Daten, im zweiten Modul *inout_mm.f* Matrix Market-Daten verarbeitet. Es wird demnach nur immer eines der beiden Module benötigt, wobei deren Aufbau sehr ähnlich ist. Auf weitere Details wird hier nicht eingegangen.

A.2 Format der FINEART Datensätze

Bei jedem Iterationsschritt werden von den FINEART-Modulen die folgenden Dateien erzeugt, die als Eingabedaten für den Gleichungslöser dienen, wobei der Projektname aus einer Projektbezeichnung *projekt* mit Angabe des Iterationsschrittes x in der Form *projekt_lx* besteht:

(i) Rechte-Hand-Seite $\vec{b}$: *projekt.dat*
Die erste Zeile dieser Datei besteht aus zwei Werten: der Anzahl der Knoten und die der Freiheitsgrade je Knoten. Das Produkt dieser beiden Zahlen ergibt die maximale Anzahl an Unbekannten, die sich durch Ausnutzung der Randbedingungen noch verringert.
In der folgenden Zahlenreihe werden die Unbekannten durchnummeriert, wobei eine Null für eine Variable steht, die wegen entsprechender Fesselung des zugehörigen Knotens nicht berechnet werden muss.

Im letzten Block stehen die Werte für die Rechte Hand Seite, deren Anzahl dem größten Eintrag in der vorigen Zahlenreihe entspricht.

(ii) Elemente der Matrix A: *gstif*
In dieser Datei sind alle Werte der Matrix A ungleich Null in binärer Darstellung abgespeichert. Sie sind der Reihe nach abgelegt, falls die Matrix spaltenweise durchlaufen wird. Diese Datei wird nach jedem Iterationsschritt überschrieben.

(iii) Positionen der Elemente ungleich Null in der Matrix A: *projekt.solv*
Hier werden zwei Zahlenblöcke abgespeichert (Abb. A.2). Der zweite Block enthält die Zeilenposition jeden Wertes aus der Datei *gstif* in der Matrix A. Die Spaltenposition wird aus dem ersten Block hergeleitet, der eine Referenz auf den zweiten darstellt. Zahl 1 bestimmt die Position des ersten Wertes von Spalte 1 in dem zweiten Zahlenblock. Alle darauf folgenden Werte bis zur Position von Zahl 2 werden ebenso dieser Spalte zugeordnet. Ab dieser Position befinden sich die Werte von Spalte 2 usw.

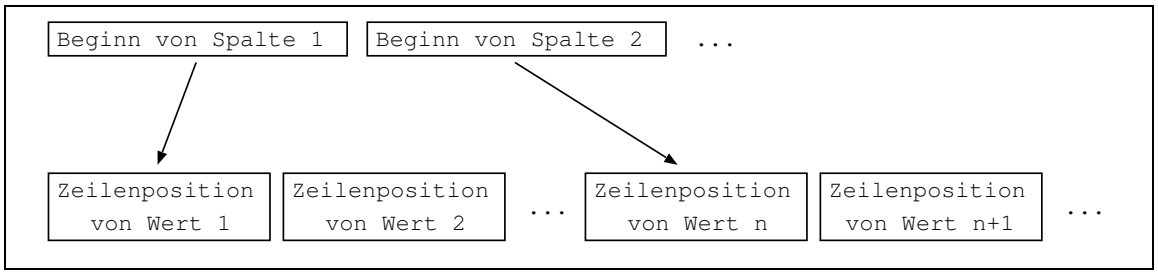


Abbildung A.2: Art der Speicherung der Matrixeinträge in der Datei *gstif*

(iv) Ergebnisvektor $\vec{x}$: *projekt.out*
In dieser Datei wird der Ergebnisvektor x des Gleichungssystems geschrieben. Er wird tabellarisch, bezogen auf das ursprüngliche Finite-Elemente-Problem, dargestellt. Zu Beginn jeder Zeile steht die Knotennummer; am Kopf jeder Spalte die Nummer des Freiheitsgrades. Die Tabelle wird mit den Werten des Ergebnisvektors x und Nullen an des Stellen, wo Gleichungen gestrichen wurden, gefüllt.

(v) Projektname *projekt* und aktueller Iterationsschritt: *project1*

Die Daten werden mit dieser „etwas komplizierten“ Methode erstellt, um eine hohe Speichereffizienz und somit auch geringe Zeit für Lese- bzw. Schreibzugriffe zu gewährleisten.

A.3 Einbindung in FINEART

Um das entwickelte Programm in FINEART zum Lösen des LGS nutzen zu können, werden entsprechende Schnittstellen bereitgestellt, realisiert durch das in Kapitel A.1 erwähnte I/O Modul. Damit können die benötigten, FINEART spezifischen Daten gelesen und Ergebnisse entsprechend ausgegeben werden.
Aus historischen Gründen geschieht der Aufruf von FINEART zunächst durch das kompilierte C-Programm *finestart*, das wiederum das Shell Script *fineart.scr* startet. Hier werden die einzelnen Module der Reihe nach ausgeführt. Um das neue Modul implementieren zu können, muss der Eintrag zum Start des Lösers durch einen Aufruf des neuen Moduls ersetzt werden.

Anhang B

RAPS-Formate

RAPS [38] ist ein interaktives Grafikprogramm zur Darstellung der Ergebnisse von Finite-Elemente-Berechnungen. Es kann Datensätze mit folgendem Aufbau verarbeiten, wie sie auch von FINEART erzeugt werden. Dabei ist die erste Zeile der Dateikopf. Die darauf folgenden Zeilen sind meist Gruppenköpfe mit Datenzeilen, die sich wiederholen können. Die führende Zeichenkette der jeweiligen Beschreibung gibt den Wertetyp, entsprechend FORTRAN-Format, an: *A* bedeutet beliebiges ASCII-Zeichen, *I* Integer und *E* Gleitkommazahl in exponentieller Schreibweise. Die folgende Zahl gibt die Anzahl an Ziffern bzw. Zeichen an. *X* steht für Leerzeichen, *#* für mehrere Einträge.

B.1 Elemente-Datei *.elemente (ASCII)

Dateikopf	
A3	Standard ist Permas 7 Kurz-Format: 'P7S'
I3	Eigenschaften des Netzes, Standard: '1'
A60	Kommentar
I6	# Elemente
Element-Gruppen-Köpfe	(werden für jede Elementgruppe wiederholt)
A8	Elementtyp
I6	Nummer der Element-Gruppe
I6	# Elemente in der Gruppe
I6	# Knoten je Element
5X	
A8	Bezeichnung der Element-Gruppe
Datenzeilen	
I7	Element-Nummern
#(I7)	Liste der Element-Knoten

B.2 Freiheitsgrade (DOF)-Datei *.freedoms (ASCII)

Dateikopf	
I5	Begrenzer, Standard: '-4444'
1X I3	Eigenschaften des Netzes, Standard: '1'
A4	'FRDM'
I3	Blöcke: 1. Block: unterdrückte DOFs 2. Block: äußere DOFs 3. Block: vorgeschriebene DOFs 4. Block: abhängige DOFs
I3	# Freiheitsgrade (maximum:6)
A56	Kommentar
I5	Begrenzer, Standard: '-4444'
Datenzeilen	(nur Knoten mit mindestens einer Randbedingung werden aufgelistet)
I6	Knoten-Nummer
#(A#,1X)	Jede Spalte ist ein Freiheitsgrad und kann den Wert 'T' (True / Randbedingung) oder 'F' (False / keine Randbedingung) haben
Letzte Zeile	
I6	Begrenzer, Standard: '-4444'

B.3 Material-Datei *.mat (ASCII)

Dateikopf	
A4	'MAT'
I3	Eigenschaften des Netzes, Standard: '1'
A60	Kommentar
I6	# Elemente
Element-Gruppen-Köpfe	(werden für jede Elementgruppe wiederholt)
A8	Elementtyp
I6	Nummer der Element-Gruppe
I6	# Elemente in der Gruppe
I6	# von Materialdaten, Standard: '6', 1. E-Modul, 2. Poisson-Zahl, 3. Dichte 4. kalthärtender Parameter, 5. Wärmeausdehnungskoeffizient, 6. Schmelztemperatur
5X	
A8	Bezeichnung der Element-Gruppe
Datenzeilen	(homogene Materialdaten innerhalb einer Element-Gruppe werden vorausgesetzt)
I6	Nummer der Element-Gruppe
#E15.7	Materialdaten für Elemente in dieser Gruppe

B.4 Geometrie-Datei *.geo (ASCII)

Dateikopf	
A4	'GEO'
I3	Eigenschaften des Netzes, Standard: '1'
A60	Kommentar
I6	# Elemente
Köpfe der Typen	(der geometrischen Daten)
I6	Typ-Nummer
I6	# Element-Gruppen dieses Typs
5X A8	Typ von geometrischem Datensatz (zurzeit nur „Shell“)
5X A8	Bezeichnung des geometrischen Datenfelds (zurzeit nur „Thick“)
Element-Gruppen-Köpfe	(werden für jede Elementgruppe wiederholt)
A8	Elementtyp
I6	Nummer der Element-Gruppe
I6	# Elemente in Gruppe
I6	# geometrischer Daten (maximum:4)
5X	
A8	Bezeichnung der Element-Gruppe
Datenzeilen	(homogene geometrische Daten innerhalb einer Element-Gruppe werden vorausgesetzt)
I6	Nummer der Element-Gruppe
#E15.7	Geometrische Daten für jeden Knoten

B.5 Datei der inkompatiblen Knoten *.mpc (ASCII)

Dateikopf	
A4	'MPC'
I3	Eigenschaften des Netzes, Standard: '1'
A60	Kommentar
I7	# inkompatibler Knoten
Köpfe der Typen	(der inkompatiblen Knoten)
I6	Typ-Nummer: 1. inkompatible Knoten von **QUAM4**-Element 2. inkompatible Knoten von **QUAM8**-Element 3. inkompatible Knoten von **QUAD4**-Element 4. inkompatible Knoten von **QUAD8**-Element 5. inkompatible Knoten von **HEXE8**-Element (noch nicht implementiert) 6. inkompatible Knoten von **HEXE20**-Element (noch nicht implementiert)
Datenzeilen	(jeder inkompatibler Knoten in einer separaten Zeile)
I6	# der führenden Knoten
#(I7)	Liste der führenden Knoten
I7	inkompatible Knoten je Nummer

B.6 Gesamtlast-Datei *.ubl (Binär)

Diese Datei enthält umgerechnete Knotenpunktkräfte aus externen Lasten (Kap. B.8.1).

B.7 Datei der Knotenpunkt-Koordinaten *.npco (Binär)

Diese Datei enthält die Koordinaten aller Knotenpunkte (Kap. B.8.1).

B.8 Binär-Dateien

B.8.1 Knoten-sortierte Dateien

Beispiele sind:
Gesamtlast-Datei *.ubl
Knotenpunkt-Koordinaten Datei*.npco
Knotenpunkt-Verschiebungen Datei*.usr
Reaktionskräfte Datei *.reak

Dateikopf

Begrenzer: ’-1111’
Dateiname (immer genau 4 Zeichen)
’1’
aktiver Spalten
’1’
Lastfall (Datei *.npco: ’1’)
(im Fall mehr als 5 Datenspalten: füge ’0’ für jede Spalte ein)

Datenzeilen (mindestens 5 Datenspalten werden geschrieben)
Knoten-Nummer
Datenspalte

Letzte Zeile

Begrenzer: ’-9999’
’FIN ’
’0’
’0’
’0’
’0’
(im Fall mehr als 5 Datenspalten: füge ’0’ für jede Spalte ein)

B.8.2 Element-sortierte Dateien

Element Spannungstensor Knoten Datei *.sigs

Element Köpfe

Begrenzer: ’-2222’
’SIGS’
’1’
’1’
aktiver Spalten, Standard: ’7’
’1’
Lastfall
Nummer der Element-Gruppe
Element-Nummer **innerhalb der Gruppe**

Datenzeilen (mindestens 8 Datenspalten werden geschrieben)
Knoten-Nummer
Datenspalten

Letzte Zeile siehe Kapitel B.8.1

Literaturverzeichnis

[1] L. ISSLER, H. RUOSS, P. HÄFELE: *Festigkeitslehre – Grundlagen*, Springer Berlin, 1995.

[2] D. GROSS, W. HAUGER, W. SCHNELL: *Technische Mechanik Band 2 – Elastostatik*, Springer Berlin, 1992.

[3] R. STEINBUCH: *Finite Elemente – Ein Einstieg*, Springer Berlin, 1998.

[4] J. ARGYRIS, H.-P. MLEJNEK: *Die Methode der finiten Elemente Band 1*, Vieweg Braunschweig, 1986.

[5] P. FRÖHLICH: *FEM-Leitfaden – Einführung und praktischer Einsatz von Finite-Element-Programmen*, Springer Berlin, 1995.

[6] Y. LIU: *Indroduction to finite element method*, University of Cincinnati, 2001. `http://urbana.mie.uc.edu/yliu/FEM-525/FEM-525.htm`

[7] K. SCHWEIZERHOF: *Finite Elemente I*, Universität Karlsruhe, 2001. `http://www.uni-karlsruhe.de/~mechanik/FE1_UEBUNG/liste.html`

[8] P. KNABNER, L. ANGERMANN: *Numerik partieller Differentialgleichungen*, Springer Berlin, 2000.

[9] I. N. BRONSTEIN, K. A. SEMENDJAJEW: *Teubner-Taschenbuch der Mathematik*, B. G. Teubner Stuttgart, 1996.

[10] K. JÄNICH: *Lineare Algebra*, Springer Berlin, 2001.

[11] J. STOER: *Numerische Mathematik 1*, Springer Berlin, 2002.

[12] G. ENGELN-MÜLLGES, F. REUTTER: *Numerik-Algorithmen mit FORTRAN 77 Programmen*, BI-Verlag Mannheim, 1993.

[13] K.-J. JOO, E. L. WILSON: *An adaptive finite element technique for structural dynamic analysis*, Computers and Structures, 30, 1988 (Seite 1319-1339).

[14] O. C. ZIENKIEWICZ, J. Z. ZHU: *Superconvergent derivative recovery techniques and a posteriori estimation in the finite element method. Part 1: A general superconvergent recovery technique. Part 2: The Zienkiewicz-Zhu a posteriori error estimator*, International Journal for Numerical Methods in Engineering, 23, 1992 (Seite 1331-1382).

[15] O. C. ZIENKIEWICZ, J. Z. ZHU: *A simple error estimator and adaptive procedure for practical engineering analysis*, International Journal for Numerical Methods in Engineering, 24, 1987 (Seite 337-357).

[16] K. WALDSCHMIDT (HRSG.): *Parallelrechner: Architekturen – Systeme – Werkzeuge*, B. G. Teubner Stuttgart, 1995.

[17] T. RAUBER, G. RÜNGER: *Parallele und verteilte Programmierung*, Springer Berlin, 2000.

[18] F. HOSSFELD, R. ESSER ET AL.: *Gekoppelte SMP-Systeme im wissenschaftlich-technischen Hochleistungsrechnen – Status und Entwicklungsbedarf*, BMBF Report '01 IR 903', Forschungszentrum Jülich GmbH, Pallas GmbH, RWTH Aachen, TU Dresden, 1999.

[19] *ZAM parallel nodes*, Forschungszentrum Jülich GmbH. `http://zampano.zam.kfa-juelich.de`

[20] *CRAY T3E-1200*, Forschungszentrum Jülich GmbH.
http://www.fz-juelich.de/nic/Supercomputer/computer-d.html

[21] H. MEUER, E. STROHMAIER, J. DONGARRA, H. D. SIMON: *Top500 Supercomputer Sites*, November 2001. http://www.top500.org

[22] Message Passing Interface Forum. *MPI: A message passing interface standard version 1.1*, University of Tennesse, Knoxville, 1995.
http://www.mpi-forum.org/docs/docs.html

[23] L. S. BLACKFORD, J. CHOI, A. CLEARY, E. D'AZEVEDO, J. DEMMEL, I. DHILLON, J. DONGARRA, S. HAMMARLING, G. HENRY, A. PETITET, K. STANLEY, D. WAKLER, R. C. WHALEY: *ScaLAPACK Users' Guide*, University of Tennesse and Oak Ridge National Laboratory, Knoxville, siam Philadelphia, 1997.
http://www.netlib.org/scalapack/slug/scalapack_slug.html

[24] J. CHOI, J. DONGARRA, S. OSTROUCHOV, A. PETITET, D. WALKER, R. C. WHALEY: *A proposal for a set of parallel basic linear algebra subprograms*, Computer Science Dept. Technical Report CS-95-292, University of Tennessee, Knoxville, 1995 (auch LAPACK Working Note #100).

[25] J. DONGARRA, R. C. WHALEY: *A user's guide to the BLACS v1.1*, Computer Science Dept. Technical Report CS-95-281, University of Tennessee, Knoxville, 1997 (auch LAPACK Working Note #94).

[26] J. CHOI, J. DEMMEL, I. DHILLON, J. DONGARRA, S. OSTROUCHOV, A. PETITET, K. STANLEY, D. WALKER, R. C. WHALEY: *ScaLAPACK: A portable linear algebra library for distributed memory computers – design issues and performance*, Computer Science Dept. Technical Report CS-95-283, University of Tennessee, Knoxville, 1995 (auch LAPACK Working Note #95).

[27] I. GUTHEIL, R. ZIMMERMANN: *Performance of Software for the full symmetric eigenproblem on CRAY T3E and T90 Systems*, Interner Bericht IB-2000-07, Forschungszentrum Jülich GmbH, 2000.

[28] J. CHOI, J. DONGARRA, S. OSTROUCHOV, A. PETITET, D. WALKER, R. C. WHALEY: *The Design and Implementation of the ScaLAPACK LU, QR and Cholesky Factorization Routines*, Computer Science Dept. Technical Report CS-97-347, University of Tennessee, Knoxville, 1997 (auch LAPACK Working Note #118).

[29] A. CLEARY, J. DONGARRA: *Implementation in ScaLAPACK of Divide and Conquer Algorithms for banded and tridiagonal linear systems*, University of Tennessee, Computer Science Technical Report, CS-97-358, 1997 (auch LAPACK Working Note #125).

[30] *Matrix Market*, National Institute of Standards and Technology.
http://math.nist.gov/MatrixMarket

[31] G. S. PALANI, NAGESH R. IYER, J. RAJASANKAR: *FINEART – Finite element analysis using adaptive refinement techniques – theory manual (version 1.2)*, Report No. CSD-MLP91-RR-01, Structural Engineering Research Centre, Madras, 2002.

[32] G. S. PALANI, NAGESH R. IYER, J. RAJASANKAR: *FINEART – Finite element analysis using adaptive refinement techniques – user manual (version 1.2)*, Report No. CSD-MLP91-RR-02, Structural Engineering Research Centre, Madras, 2002.

[33] G. S. PALANI, NAGESH R. IYER, J. RAJASANKAR: *FINEART – Finite element analysis using adaptive refinement techniques – programming manual (version 1.2)*, Report No. CSD-MLP91-RR-03, Structural Engineering Research Centre, Madras, 2002.

[34] T. SKALICKY: *LASpack – a package for solving large sparse systems of linear equations*, 1996.
http://www.tu-dresden.de/mwism/skalicky/laspack/laspack.html

[35] J. RAJASANKAR, NAGESH R. IYER, T. V. S. R. APPA RAO: *h-adaptive refinement of finite element meshes*, Report No. SST-RR-98-06, Structural Engineering Research Centre, Madras, 1998.

[36] *PERMAS Online Manuals*, PERMAS Version 8, INTES Publication, Stuttgart 2000.

[37] *Quad-Gen 2.5 - automatic quadrilateral mesh generator for multiply-connected domains*, Computational Mechanics Australia, 2001.
`http://www.users.bigpond.com/comecau/Quadgen.htm`

[38] D. KOSCHMIEDER: *RAPS – Finite-Element-Postprozessor*, Forschungszentrum Jülich GmbH.
`http://www.fz-juelich.de/vislab/software/raps/raps_g.html`